NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

CODE OPTIMIZATION FOR THE CHOI-WILLIAMS DISTRIBUTION FOR ELINT APPLICATIONS

by

Kenneth Barry Hollinger

December 2009

Thesis Co-Advisors: Douglas J. Fouts
Phillip E. Pace

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 2009	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE Code Optimization for the Choi-Williams Algorithm for ELINT Applications	5. FUNDING NUMBERS
6. AUTHOR(S) Kenneth Barry Hollinger	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Center for Joint Services Electronic Warfare Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research Code 312 Washington D.C.	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

This thesis investigates optimizing the speed of computation for computing the Choi-Williams distribution. The Choi-Williams distribution is a way of simultaneously representing a signal in both the time and frequency domains in a fashion that makes it possible to extract the waveform parameters of the signal. The Choi-Williams distribution is particularly useful for analyzing low probability of intercept signals for electronic intelligence applications. The usefulness of the distribution is directly correlated to the speed of computation. This thesis examines methods in which the Choi-Williams distribution can be modified to increase the speed of computation while still maintaining its ability to provide a clear picture of the signal characteristics. By eliminating the computation of near zero terms of the Choi-Williams kernel function, the speed of computation can be increased dramatically while still preserving, and improving, the time-frequency characteristics. The optimizations developed in this thesis reduced the time to compute a 512 sample CWD from 6.9 seconds, to 0.0466 seconds on an Intel chip, Linux based PC—an increase in speed of 147X.

14. SUBJECT TERMS Choi-Williams Distribution, Signal Processing, Algorithm Optimization, C programming, Low Probability of Intercept (LPI), Radar detection, Radar classification	15. NUMBER OF PAGES 98
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

CODE OPTIMIZATION FOR THE CHOI-WILLIAMS DISTRIBUTION FOR ELINT APPLICATIONS

Kenneth B. Hollinger
Captain, United States Marine Corps
B.S., University of Idaho, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2009

Author: Kenneth Barry Hollinger

Approved by: Douglas J. Fouts
Co-Advisor

Phillip E. Pace
Co-Advisor

Jeffrey B. Knorr
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis investigates optimizing the speed of computation for computing the Choi-Williams distribution. The Choi-Williams distribution is a way of simultaneously representing a signal in both the time and frequency domains in a fashion that makes it possible to extract the waveform parameters of the signal. The Choi-Williams distribution is particularly useful for analyzing low probability of intercept signals for electronic intelligence applications. The usefulness of the distribution is directly correlated to the speed of computation. This thesis examines methods in which the Choi-Williams distribution can be modified to increase the speed of computation while still maintaining its ability to provide a clear picture of the signal characteristics. By eliminating the computation of near zero terms of the Choi-Williams kernel function, the speed of computation can be increased dramatically while still preserving, and improving, the time-frequency characteristics. The optimizations developed in this thesis reduced the time to compute a 512 sample CWD from 6.9 seconds, to 0.0466 seconds on an Intel chip, Linux based PC—an increase in speed of 147X.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Low Probability of Intercept (LPI) emitters are the next evolution in radar technology and represent a growing threat to operating forces, civilian targets, and national security. The operational implementation of time-frequency signal processing techniques such as the Choi-Williams distribution (CWD) are useful for the detection and classification of these threat waveforms. The techniques, however, are currently limited due to non-real-time computational constraints. Developing a system that can detect and classify LPI emitters in real time is the first step in developing countermeasures to this growing threat.

This thesis investigates developing a highly optimized algorithm for computing the Choi-Williams distribution. Using a C code implementation as a benchmark, an increase of over 100X in speed of computation was achieved. It is shown that the optimizations also produce a clearer and more accurate picture of the input signal characteristics. The algorithm developed for this thesis can be ported to any platform. Many of the optimizations developed can be applied to the computation of the Wigner-Ville distribution as well. This work is highly applicable in the effort to develop LPI emitter signal detection and classification techniques.

There are currently several signal processing algorithms, which are able to provide time/frequency representations of an incoming signal. These signal processing techniques are useful tools in the detection and classification of LPI emitters; however, the long processing time it takes to use these techniques prevents these techniques from being used in a real time system. The goal then is to decrease the processing time to compute these algorithms.

The CWD is notable in its ability to provide a clear, human-readable, time-frequency picture of an LPI signal that is simple to classify. The ability to compute the Choi-Williams algorithm in real or near real time will be a significant contribution in developing countermeasures to weapons systems utilizing LPI emitters.

Several optimizations are presented which can be used to modify the Boasash algorithm, reducing the complexity and number of computations made. The cumulative effect of utilizing these optimizations produces a hundred fold increase in speed of computation of the Choi-Williams distribution. Some of the optimizations made can also be used to increase the speed at which the Wigner-Ville distribution (another signal processing algorithm) can be computed. These optimizations also have the unintended effect of providing a clearer and more accurate picture of an LPI signal.

To compare the final code (`pipe.c`) to the original code (`choi.c` from [5]) five test runs were conducted for both the compiler un-optimized and compiler-optimized versions of both programs. The trial runs were conducted on an Intel chip, Linux based PC. Table 1 shows the results.

	From [5]		**From this work**	
	choi.c	**choi.c (compiler optimized)**	**pipe.c**	**pipe.c (compiler optimized)**
Trail 1	46.81	6.860	0.05442	0.04699
Trial 2	46.51	6.887	0.05445	0.04655
Trial 3	46.50	6.852	0.05457	0.04640
Trial 4	46.23	6.872	0.05470	0.04631
Trial 5	46.49	6.824	0.05426	0.04661
Average	**46.51**	**6.859**	**.05448**	**.046572**

Table 1. Time in seconds of trial runs.

The optimized version of the code produced an 854X increase in speed over the original code. The optimized version of the code that was compiled using an optimizing compiler produced a 147X increase in speed over the compiler-optimized version of the original code.

LIST OF ABBREVIATIONS, ACRONYMS, AND SYMBOLS

BFM	Butterfly Machine
CWD	Choi-Williams Distribution
DFT	Discrete Fourier Transform
ELINT	Electronic Intelligence
EW	Electronic Warfare
FFT	Fast Fourier Transform
FMCW	Frequency Modulated Continuous Wave
FPGA	Field Programmable Gate Arrays
LPI	Low Probability of Intercept
N	Number of Samples
NPS	Naval Postgraduate School
SNR	Signal To Noise Ratio
WVD	Wigner-Ville Distribution

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Many thanks to Professor Fouts in allowing me a free hand in the development of this thesis.

Many thanks to Professor Pace for his unwavering faith in my ability to actually accomplish something.

Many thanks to Professors Borges and Daughtry for teaching me the math skills necessary for this thesis.

Many thanks to Professors Loomis, Jenn, Fargues, Butler, and Breitenbach for acting as sounding boards.

Many thanks to my son Gage for helping me decompress by battling Murlocs.

Many thanks to my daughter, Haley, for listening to me prattle on about my thesis.

Many thanks to Wes Simon for helping me talk to computers.

Many thanks to Mr. Briscoe at the NEX sandwich shop for his culinary masterpieces: the bacon breakfast burrito, and the Briscoe's Sockit-to-me sandwich.

This work is supported by the Office of Naval Research Code 312, Washington, D.C.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. CHOI-WILLIAMS TIME-FREQUENCY DISTRIBUTION FOR ELECTRONIC WARFARE

Low Probability of Intercept (LPI) emitters are the next evolution in radar technology and represent a growing threat to operating forces, civilian targets, and national security. The development of techniques to automatically detect and classify LPI emitters is of extremely high importance. Currently, US and allied electronic warfare (EW) systems cannot detect or classify most of the new types of LPI radar and communication systems that are currently being developed and deployed by adversary nations. LPI systems utilize techniques such as frequency hopping and direct sequence spread spectrum to avoid detection. As more and more of these systems are deployed by potential adversaries, U.S. and allied forces will be at greater and greater risk from high-speed, sea-skimming, anti-ship cruise missiles and other weapons.

There are currently several signal processing algorithms that are able to provide time-frequency representations of an incoming signal. These signal processing techniques are useful tools in the detection and classification of LPI emitters; however, the long processing time it takes to use these techniques prevents these techniques from being used in a real time system. The goal then is to decrease the processing time to compute these algorithms.

The Choi-Williams distribution (CWD) [1] is one of Cohen's generalized class of time-frequency distributions [2], which also includes the Wigner-Ville distribution and the spectrogram. Cohen's time-frequency distributions have been the focus of extensive research and study. The CWD uses an exponential kernel to reduce the magnitude of cross terms. This is an improvement over the Wigner-Ville distribution, which simply has a kernel function of one and produces large cross terms that can obscure the real signal. However, much of the research done to improve the speed of computation of the Wigner-Ville distribution is applicable to computing the CWD, as well as other time-frequency

distributions based on Cohen's generalized class. In particular the efficient algorithm developed by Boasash and Black [3], has been key in an effort to improve the speed of computation of the CWD using reconfigurable computers [4], [5]. Other research has been done to compute the CWD using parallel processing [6], [7]. Cardoso et al. [6], computed the CWD using a parallel implementation on a transputer platform with five processors. They were able to compute the CWD for 1024 samples in 20.96 ms. In [7], Barry used matrix manipulation techniques to provide an intuitive approach that, when combined with parallel processing, will improve the processing speed to allow real-time calculations of the CWD.

Other efforts to improve the computation speed include the use of instantaneous frequency [8], and the fast Hartley transform [9] in lieu of a Fast Fourier Transform (FFT). In [8], Jones and Boasash investigated the spread of a signal about its instantaneous frequency for several common time-frequency distributions. Their efforts led to an adaptive algorithm that may be used to improve the time-frequency resolution of multicomponent signals. In [9], Narayanan and Prabhu found that computing Wigner-Ville distribution using the Fast Hartley method was much faster than using an FFT method. This due to considerably less multiplications and additions needed to compute the Fast Hartley method.

Additional research in this field has been conducted in comparing the resolution of various time-frequency distributions [10], and the development of new tools for the interpretation and quantitative comparisons of high-resolution time-frequency distributions [11]. In [10], it was found that the Choi-Williams distribution is the most attractive for signals in which all components have constant frequency content. In [11], Cunningham and Williams introduce some new tools for the interpretation and quantitative comparison of high-resolution time-frequency distributions.

The CWD is notable in its ability to provide a clear, visual, time frequency picture of an LPI signal that is simple to classify. The ability to compute the Choi-

Williams algorithm in real- or near real-time will be a significant contribution in developing intercept receivers for LPI signal detection.

B. RESEARCH OBJECTIVES

The intent of this thesis work was to improve the speed with which the CWD can be computed on the SRC-6 reconfigurable supercomputer. Some initial work on this has been done [4], with limited results. The approach taken was to first study what computations were being made by the computer, then examine how to best format the computations to take advantage of the SRC-6's reprogrammable hardware.

C. APPROACH AND PRINCIPLE CONTRIBUTIONS

In studying the algorithm used (an implementation of the Boasash algorithm), it was discovered that many multiplications by zero were taking place. The algorithm was rewritten to eliminate these multiplications. Also, it was discovered that in the original algorithm, many of the same calculations were being done multiple times, such as computing of the weighting function (which never changes) N times where N is the number of signal samples.

Next, it was discovered that the conjugate symmetry of the distribution required only half of the distribution to be computed, then copied into the other half with a change of sign for the imaginary portion of the complex number. Eventually a method was found to fill the other half of the distribution with zeros. It was at this time that other ways of representing the kernel (weighting) function were investigated to see if using powers of two instead of the exponential function would allow easier computation. This manipulation was determined to not be useful for the C programming language, but was included in this thesis because it could be useful when developing a hardware solution to these algorithms.

When looking at three-dimensional plots of the kernel (weighting) function, it was noted that a very high proportion of the weights were near zero, and that

these weights contributed very little to the final result of the computation. A “cut and slice” method was developed to test various ways of stripping out computations that, while taking as much time as the other computations, would make insignificant contributions to the final result of the distribution. It was found that only a fraction of the computations need be computed to still produce excellent results. These new modifications were coded into the original C code and very significant speed gains were made.

It was also found that stripping out the low (near zero) weighted contributions to the sum would make it possible to reduce the number of computations needed to compute the FFT. An algorithm was developed for the generalized Butterfly Machine (BFM), which reduced the layers of BFMs needed for the FFT. This algorithm is directly applicable to the computation of zero-padded FFTs. The FFT from the benchmark code [5], written by Prof. Breitenbach, was modified with his permission to incorporate these new changes and the total computation time of the CWD was again significantly reduced.

Next, an analysis was done to objectively determine the affects of eliminating the computation of near zero terms in the weighted summations. As would be expected, there is a strong correlation between the number of computations eliminated and the departure in results from the unaltered CWD. However, in many cases, the time-frequency picture appears smoother in the CWD implemented using the “cut and slice” method than using the unaltered CWD. For a signal that is changing frequencies with time (such as an LPI signal), contributions to the frequency at time t by a sample taken at time $t+\tau$ are increasingly irrelevant as τ increases, and in fact degrade the time frequency picture. With further analysis it was found that the “cut and slice” method greatly reduces the contribution of these irrelevant samples—reducing clutter that obscures the true signal

Finally, all of the modifications were coded into the C programming language in the most efficient manner as possible and the program was written such that it could be easily ported to a pipelined system where each new sample

produces a new CWD. The new code was run using the Upperman code [5] as a benchmark. The new code ran 854 times as fast as the old code when run using no compiler optimizations and 147 times faster when both sets of code were optimized by the compiler. The optimized version of the code was able to compute the CWD for a 512 sample signal in 46.6 ms on an Intel chip, Linux based PC.

The approximate factor of 6 increase in speed improvement for the compiler un-optimized version can be attributed to standard computer engineering coding principles such as loop unrolling, minimized decision making, and the reduction of redundant computations. The new code is highly optimized to be ported to a hardware implementation of the CWD, as well as a reconfigurable computer such as the SRC-6.

D. THESIS OUTLINE

Chapter II of this thesis provides a step by step implementation of the Boasash algorithm, which is used to compute the CWD. It provides a breakdown of the equations of the CWD, and provides a step by step breakdown of how the distribution can be computed using the Boasash algorithm. It also provides graphical representations of the affects of each stage of the computation, providing a clear picture of what is happening to the signal at each stage of computation.

Chapter III steps through several modifications that can be made to the algorithm. Each section shows the math behind the modification, how each modification reduces the number of computations in the algorithm, and the result (if any) on the final result of the distribution.

Chapter IV examines the cumulative effect of all the optimizations on the distribution, documenting both the error produced by the optimizations and the increase in processing speed achieved. It is shown that, in addition to the hundred-fold increase in processing speed accomplished, the optimized computing algorithm actually produces a more accurate representation of the LPI signal. Chapter V provides the conclusions that can be drawn from the work accomplished in this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

II. COMPUTATION METHOD

A. CHOI-WILLIAMS DISTRIBUTION

The CWD is one of a class of time-frequency distributions introduced by Cohen. In discrete form, the CWD can be expressed as

$$CWD_x(\ell,\omega) = 2\sum_{n=-\infty}^{\infty} W(n)S(\ell,n)e^{-j2\omega n} \tag{2.1}$$

where

$$S(\ell,n) = \sum_{\mu=-\infty}^{\infty} W(\mu)\frac{1}{\sqrt{4\pi n^2/\sigma}}e^{-\frac{(\mu-\ell)^2}{4n^2/\sigma}}x(\mu+n)x^*(\mu-n) \tag{2.2}$$

and $W(\mu)$ is a uniform window that has a value of one for the range $-M/2$ through $M/2$, and $W(n)$ is a uniform window that has a value of one for the range $-N/2$ through $N/2$ and is zero elsewhere [1].

The above equations can be modified in order to tailor them to allow the use of a standard FFT (Fast Fourier Transform) in calculating the distribution. For the work described in this thesis, all examples will use data sets of N samples where N is a power of two. Setting the summation windows over n and μ to go from $-N/2$ to $N/2-1$, and having ℓ realize the same range, will cover all non-zero values in the distribution and will enable the use of the FFT. This will be clearer in the following analysis.

Without loss of generality, the N samples are labeled $x\left(-N/2\right)$ through $x\left(N/2-1\right)$. Equation (2.2) can now be expressed as

$$S(\ell,n) = \sum_{\mu=-N/2}^{N/2-1} \phi(\mu,\ell,n)A\left(\mu,n\right) \tag{2.3}$$

where

$$A\left(\mu,n\right) = x(\mu+n)x^*(\mu-n) \tag{2.4}$$

and

$$\phi(\mu,\ell,n)=\frac{1}{\sqrt{4\pi n^2/\sigma}}e^{-\frac{(\mu-\ell)^2}{4n^2/\sigma}} \tag{2.5}$$

Note that (2.5) is undefined for $n=0$. Using the substitutions

$$\kappa=\mu\frac{1}{\sqrt{4n^2/\sigma}} \qquad \Rightarrow \qquad d\kappa=\frac{1}{\sqrt{4n^2/\sigma}}d\mu \qquad \Rightarrow \qquad d\mu=\sqrt{4n^2/\sigma}d\kappa$$

note that

$$\int_{\mu=-\infty}^{\infty}\frac{1}{\sqrt{4\pi n^2/\sigma}}e^{-\frac{(\mu-\ell)^2}{4n^2/\sigma}}d\mu=\int_{\mu=-\infty}^{\infty}\frac{1}{\sqrt{4\pi n^2/\sigma}}e^{-\frac{\mu^2}{4n^2/\sigma}}d\mu=\int_{\kappa=-\infty}^{\infty}\frac{\sqrt{4n^2/\sigma}}{\sqrt{4\pi n^2/\sigma}}e^{-\kappa^2}d\kappa$$

$$=\frac{1}{\sqrt{\pi}}2\int_{\kappa=0}^{\infty}e^{-\kappa^2}dk$$

Using the identity $\int_{x=0}^{\infty}e^{x^2}\,dx=\frac{\sqrt{\pi}}{2}$ [12]:

$$\frac{1}{\sqrt{\pi}}2\int_{\kappa=0}^{\infty}e^{-\kappa^2}\,dk=\frac{2}{\sqrt{\pi}}\frac{\sqrt{\pi}}{2}=1$$

This shows that (2.5) is a Gaussian distribution over μ, which has a variance of zero for $\mu=\ell$ when $n=0$. Using the sifting property:

$$\phi(\mu,\ell,n=0)=\begin{cases}1 & \mu=\ell\\ 0 & \mu\neq\ell\end{cases}$$

If we define the function $S'(\ell,n)$ to be (this is a generalization of the equation used in the Boasash algorithm),

$$S'(\ell,n)=\begin{cases}S(\ell,n) & 0\le n\le N/2-1\\ 0 & n=N/2\\ S(\ell,n-N) & N/2+1\le n\le N-1\end{cases}$$

In this way, $S(\ell,n)$ for $n=-N/2$ through $N/2-1$ is mapped to $S'(\ell,n)$ for $n=0$ through $N-1$. So (2.1) becomes:

$$CWD_x(\ell,\omega) = 2\sum_{n=0}^{N-1} S'(\ell,n)e^{-j2\omega n}$$

Finally, this representation can be modified to fit the standard Fast Fourier Transform (FFT) by making the substitution: $\omega = \pi k / N$ thus 2.1 becomes

$$CWD_x(\ell,\frac{\pi k}{N}) = 2\sum_{n=0}^{N-1} S'(\ell,n)e^{-j2\pi k\frac{n}{N}} \qquad k = 0,1,...,N-1 \tag{2.6}$$

Note that transforming 2.1 into an FFT equation results in frequencies that are double what they initially were. This effect can be nulled by simply halving the values of the frequency axis in the plotted representations of the signal.

This section has shown how the discrete CWD is modified in order to compute the distribution for an N sample window using an FFT. The next section breaks down the Boasash algorithm into steps and shows the results of performing each step on an example input signal.

B. COMPUTING THE DISTRIBUTION

This section details how to compute the CWD using the equations in the previous section and shows an example of the resultant output for a Costas frequency hopping signal with a signal to noise ratio (SNR) of 0 dB sampled $N = 512$ times (Figure 1). The Appendix shows the details for the signals used in this thesis.

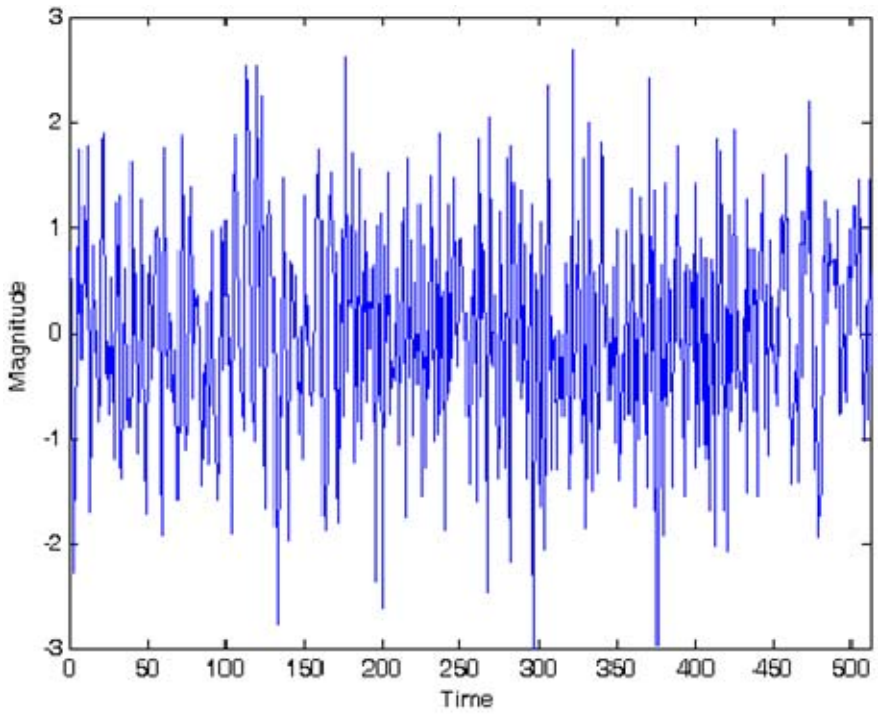


Figure 1. 512 samples of Costas signal (real portion).

The algorithm used to compute the CWD for a signal sampled N times can be expressed in the following manner. First the N samples are stored in an array. This gives us $x(k)$ where

$$x(k) = \begin{cases} \text{sample at time k} & -N/2 \le k \le N/2 - 1 \\ 0 & \textit{elsewhere} \end{cases} \tag{2.7}$$

Next, the function $A(\mu, n)$ can be written to an N by N array. For illustration purposes, Figure 2 shows an example of how $A(\mu, n)$ for an $N = 8$ sample signal would be arrayed.

$\mu \setminus n$	0	1	2	3	-4	-3	-2	-1
3	x(3)·x(3)*	x(4)·x(2)*	x(5)·x(1)*	x(6)·x(0)*	x(-1)·x(7)*	x(0)·x(6)*	x(1)·x(5)*	x(2)·x(4)*
2	x(2)·x(2)*	x(3)·x(1)*	x(4)·x(0)*	x(5)·x(-1)*	x(-2)·x(6)*	x(-1)·x(5)*	x(0)·x(4)*	x(1)·x(3)*
1	x(1)·x(1)*	x(2)·x(0)*	x(3)·x(-1)*	x(4)·x(-2)*	x(-3)·x(5)*	x(-2)·x(4)*	X(-1)·x(3)*	x(0)·x(2)*
0	x(0)·x(0)*	x(1)·x(-1)*	x(2)·x(-2)*	x(3)·x(-3)*	x(-4)·x(4)*	x(-3)·x(3)*	x(-2)·x(2)*	x(-1)·x(1)*
-1	x(-1)·x(-1)*	x(0)·x(-2)*	x(1)·x(-3)*	x(2)·x(-4)*	x(-5)·x(3)*	x(-4)·x(2)*	x(-3)·x(1)*	x(-2)·x(0)*
-2	x(-2)·x(-2)*	x(-1)·x(-3)*	x(0)·x(-4)*	x(1)·x(-5)*	x(-6)·x(2)*	x(-5)·x(1)*	x(-4)·x(0)*	x(-3)·x(-1)*
-3	x(-3)·x(-3)*	x(-2)·x(-4)*	x(-1)·x(-5)*	x(0)·x(-6)*	x(-7)·x(1)*	x(-6)·x(0)*	x(-5)·x(-1)*	x(-4)·x(-2)*
-4	x(-4)·x(-4)*	x(-3)·x(-5)*	x(-2)·x(-6)*	x(-1)·x(-7)*	x(-8)·x(0)*	x(-7)·x(-1)*	x(-6)·x(-2)*	x(-5)·x(-3)*

Figure 2. $A(\mu, n)$ for $N = 8$

Note that in Figure 2, several of the boxes are grey. These boxes are highlighted because one or both terms in the box falls outside of the sample window ($x(-4)$ through $x(3)$) and the result of the complex multiply will be zero.

Figure 3 shows the absolute values of what becomes of our Costas signal once it has been processed in the same manner and mapped into a 512 by 512 array. Note, that $A(\mu, n)$ is not dependent upon the parameter ℓ. This array will be used for all subsequent calculations and need only be computed once.

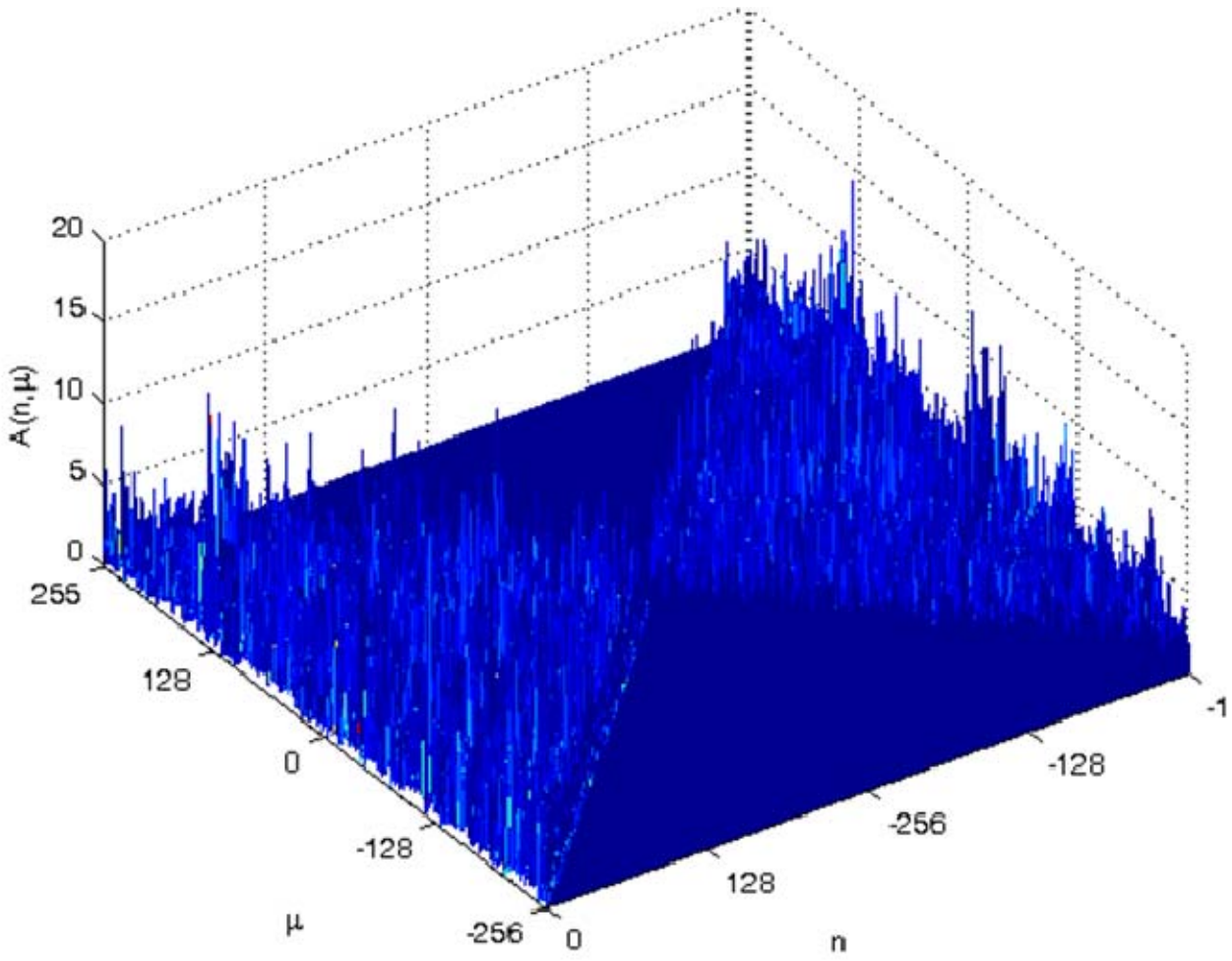


Figure 3. $A(\mu, n)$ of Costas signal.

The remaining steps in the process of computing the CWD will be repeated for all ℓ. For this example, $-256 \le \ell \le 255$. The following figures are used to illustrate the algorithm and are all computed with $\ell = 0$, a time in the middle of the sampled signal.

First, a matrix is created to represent the kernel function $\phi(\mu, \ell, n)$ shown in (2.5). Figure 4 shows the result for all μ and n, with $\sigma = 1$, and $\ell = 0$. The value of σ (sigma) is a constant, and will be discussed in more detail later.

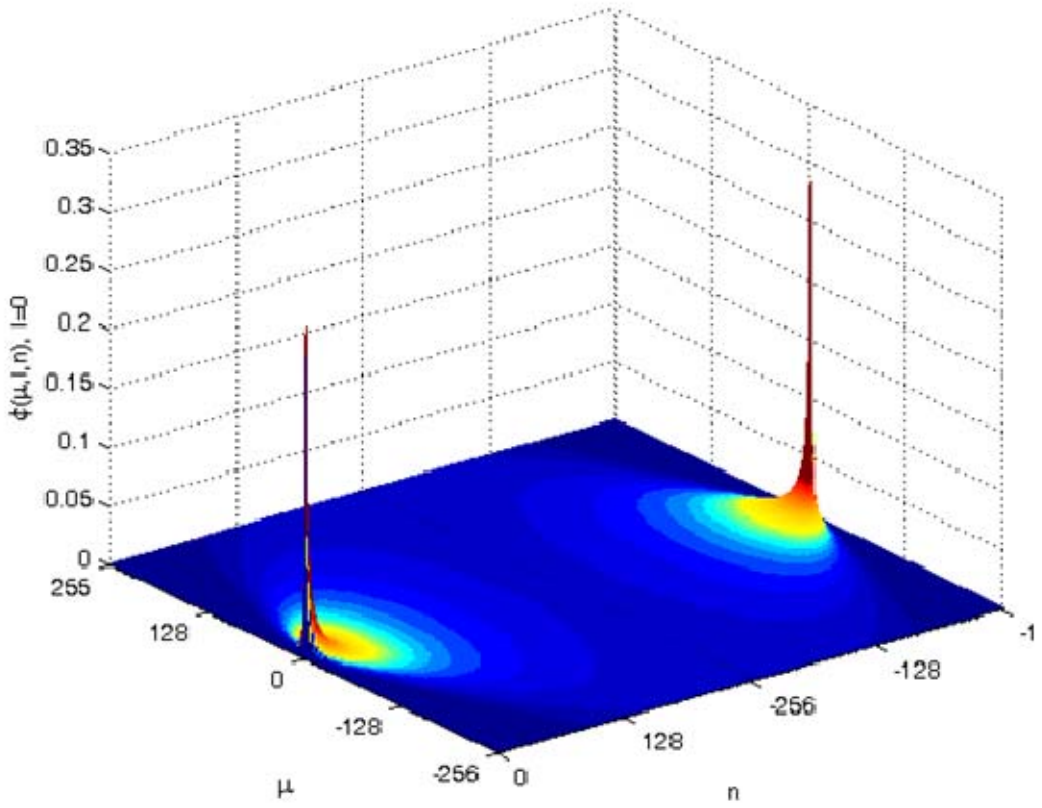


Figure 4. Choi-Williams kernel function $\ell = 0$, $N = 512$.

Next, the array is weighted (multiplied) by the kernel function, producing an array that represents $\phi(\mu, \ell, n) A(\mu, n)$ for all n and μ for the current iteration, of ℓ. Figure (5) shows the kernel function for $\ell = 0$.

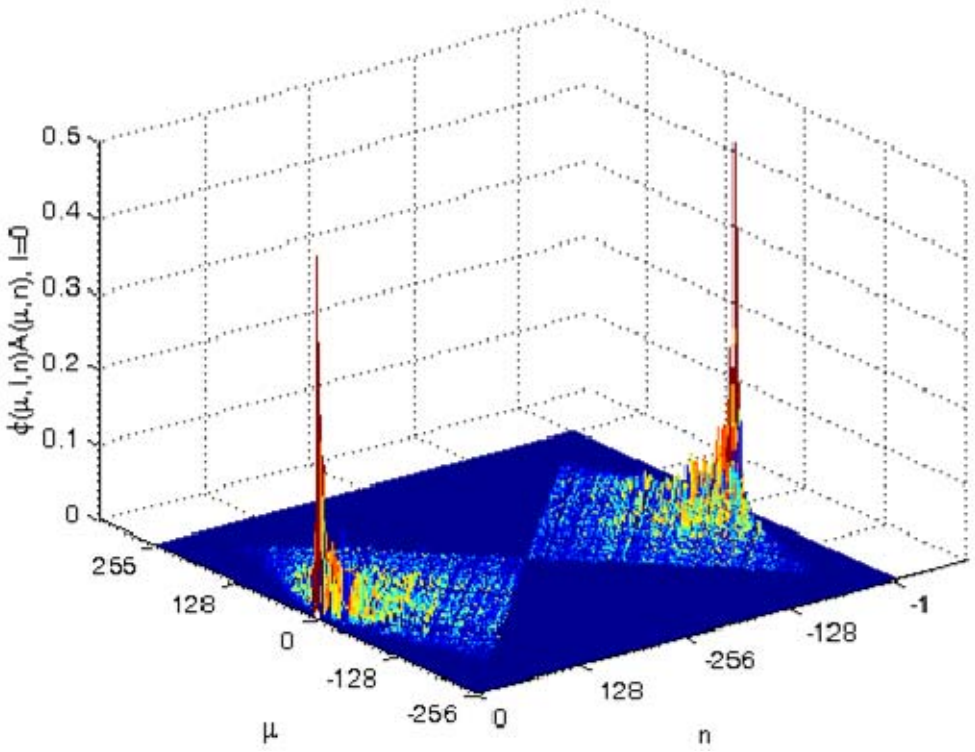


Figure 5. $\phi(\mu,\ell,n)A(\mu,n)$ for $\ell=0$.

Next, each column is summed producing a 1 x N array representing

$$S'(\ell,n)=\sum_{\mu=-N/2}^{N/2-1}\phi(\mu,\ell,n)A(\mu,n)$$

for $\ell=0$. Figure 6 shows the $S'(\ell,n)$ for $\ell=0$ (the results of the summation over μ).

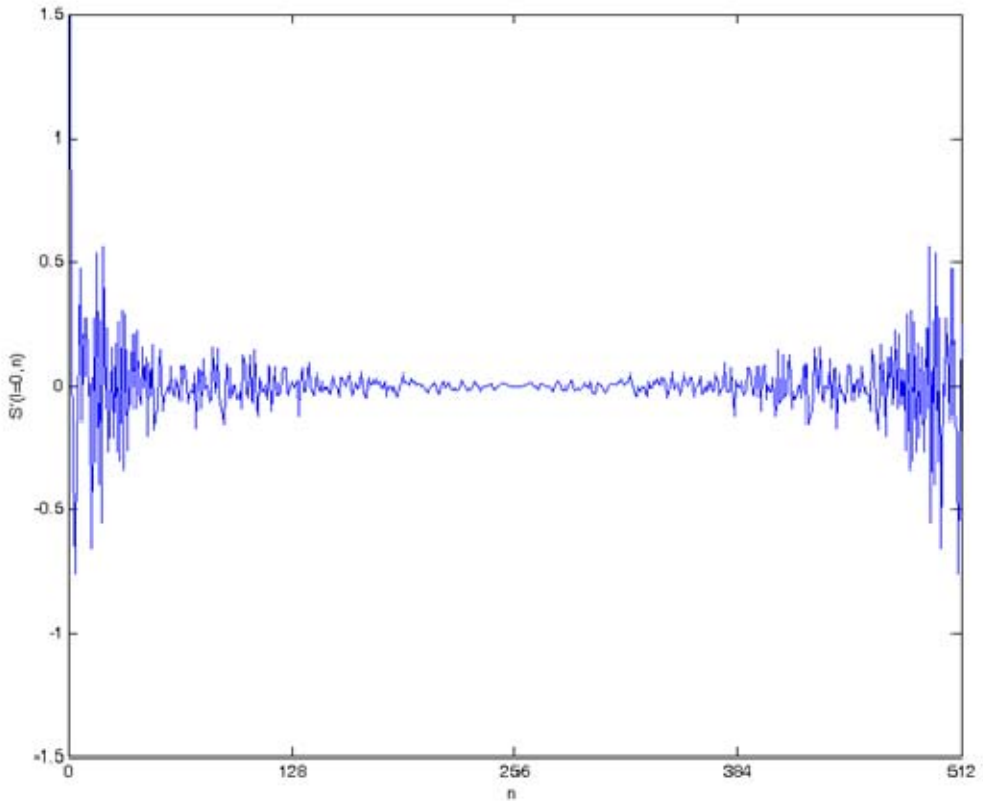


Figure 6. $S^{'}(\ell, n)$ for $\ell = 0$.

Finally, an FFT is used to compute the frequency for the current ℓ and saved to an array that represents

$$CWD_x(\ell, \frac{\pi k}{N}) = 2\sum_{n=0}^{N-1} S^{'}(\ell, n) e^{-j2\pi k \frac{n}{N}}$$

for the current ℓ. Figure 7 shows the result of the FFT for $\ell = 0$. Figure 8 depicts the entire time-frequency distribution for all ℓ. The red line shows the proper place in the time-frequency plot for the $\ell = 0$ calculation. Notice that the time-frequency plot shows the frequency crossing the $\ell = 0$ line to be 4000, which reflects the frequency shown in Figure 7. The time-frequency plot is a top down view of the results for all ℓ, with colors mapped so that high returns show in red and low returns in blue.

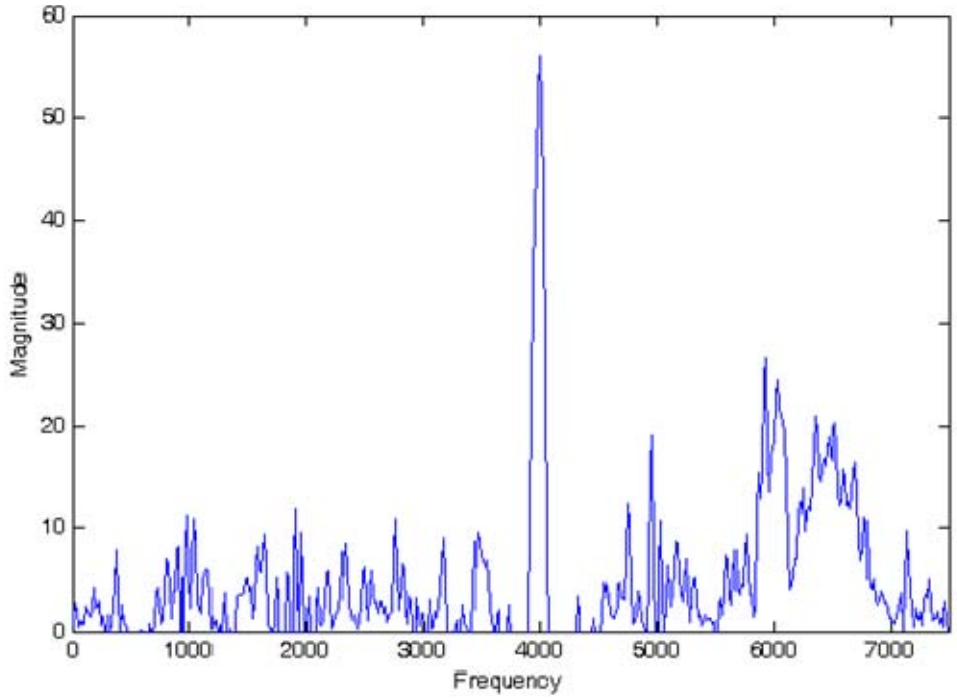


Figure 7. The discrete CWD for $\ell = 0$.

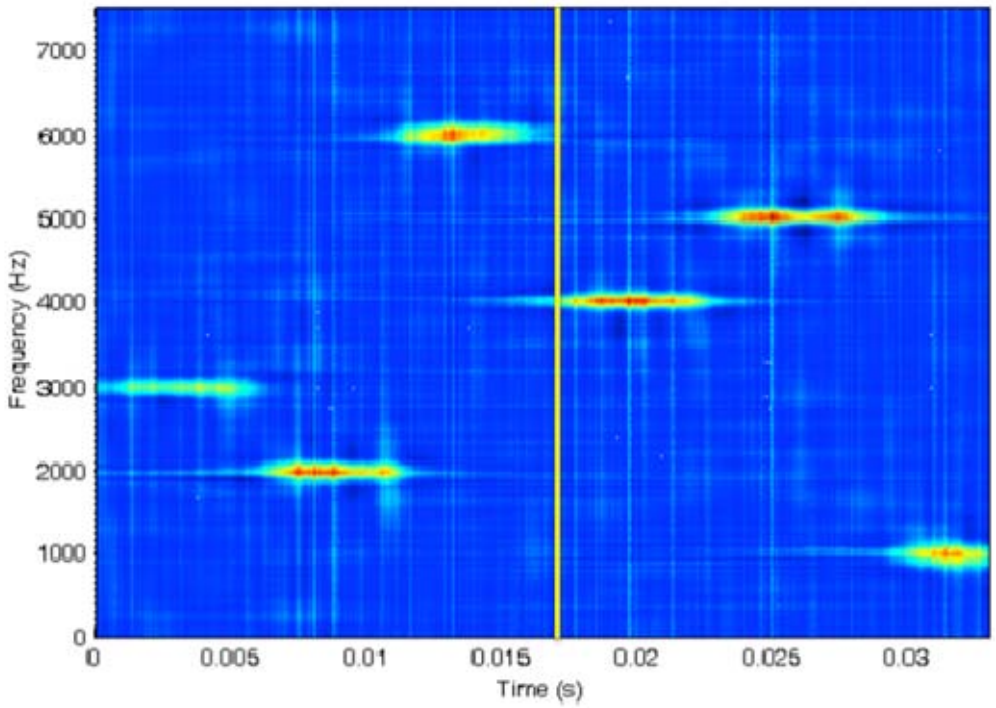


Figure 8. Yellow line shows location of CWD for $\ell = 0$ in the time-frequency distribution.

When this algorithm is computed for all ℓ, the resulting array can be plotted representing frequency as a function of time. The resulting plot, Figure 9, provides a picture of the signal that is simple to analyze. The time-frequency plot is a top down view of the results for all ℓ, with colors mapped so that high returns show in red and low returns in blue.

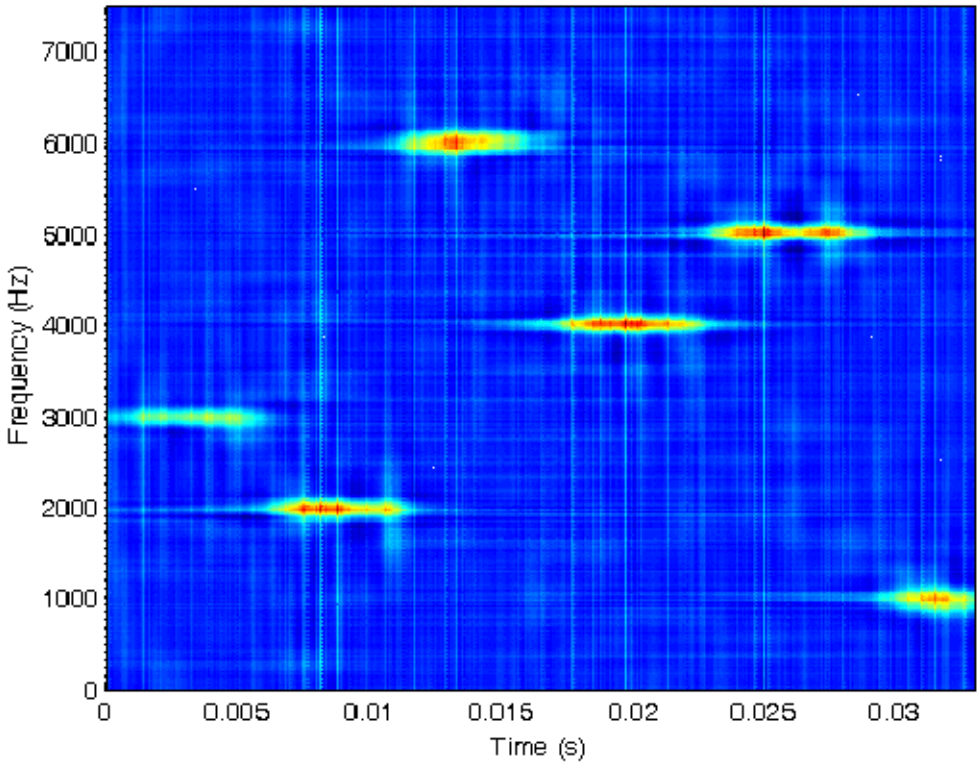
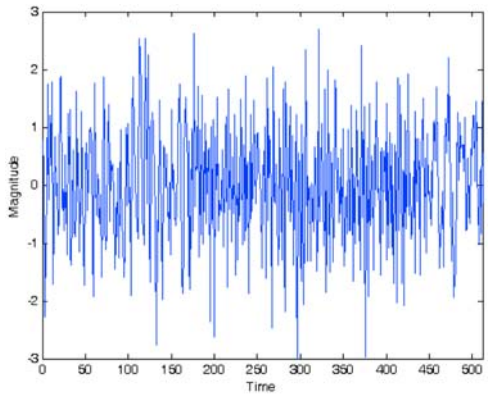
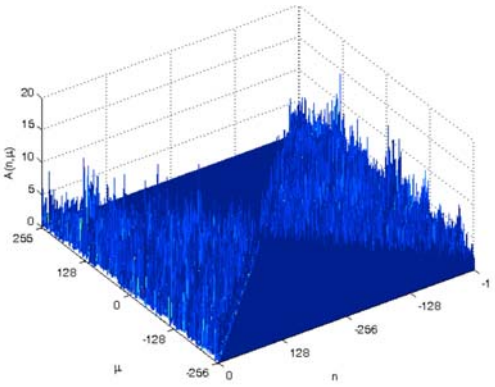


Figure 9. Plot of Choi-Williams processed Costas signal, $SNR = 0\,\mathrm{dB}$.

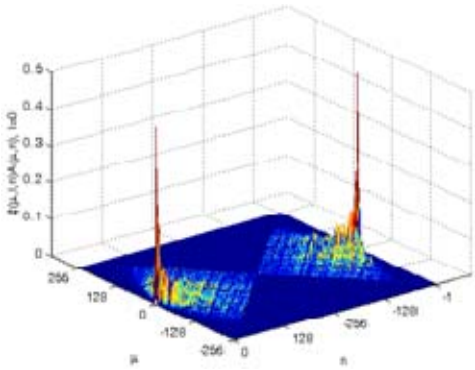
The algorithm expressed in this section was implemented in C code [4], [5]. The average time to complete this algorithm for an $N = 512$ sample was approximately 46 seconds. This time was reduced to approximately 6 seconds using compiler optimizations. Figure 10 shows a summary of this algorithm. The next chapter steps through the optimizations that were used to accomplish this.



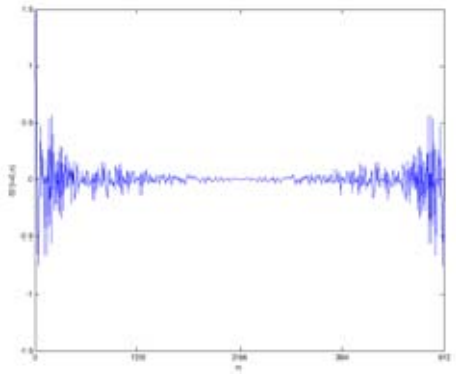
1. The signal is sampled N times.



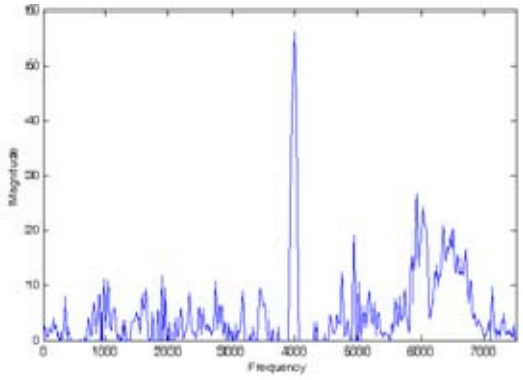
2. $A(\mu,n)$ is computed.



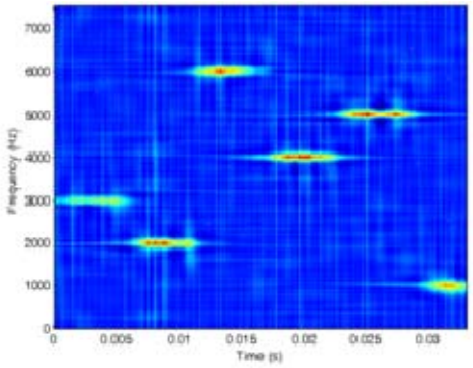
3. $A(\mu,n)$ is weighted by the kernel function.



4. The result is summed over μ.



5. Then fed through an FFT.



6. Steps 3 through 5 are computed for all ℓ.

Figure 10. A summary of the steps to compute the CWD.

III. OPTIMIZATIONS FOR FASTER COMPUTATION

A. ELIMINATE COMPUTATIONS THAT WILL GIVE ZERO AS A RESULT

Equation (2.7) states that all values for $x(k)$ outside the range $-N/2 \le k \le N/2 - 1$ are equal to zero. The resulting array in Figure 11 is functionally equivalent to the array in Figure 2. This reduces the number of computations for the array by a factor of 2.

$\mu \setminus n$	0	1	2	3	-4	-3	-2	-1
3	x(3)·x(3)*	0	0	0	0	0	0	0
2	x(2)·x(2)*	x(3)·x(1)*	0	0	0	0	0	x(1)·x(3)*
1	x(1)·x(1)*	x(2)·x(0)*	x(3)·x(-1)*	0	0	0	X(-1)·x(3)*	x(0)·x(2)*
0	x(0)·x(0)*	x(1)·x(-1)*	x(2)·x(-2)*	x(3)·x(-3)*	0	x(-3)·x(3)*	x(-2)·x(2)*	x(-1)·x(1)*
-1	x(-1)·x(-1)*	x(0)·x(-2)*	x(1)·x(-3)*	x(2)·x(-4)*	0	x(-4)·x(2)*	x(-3)·x(1)*	x(-2)·x(0)*
-2	x(-2)·x(-2)*	x(-1)·x(-3)*	x(0)·x(-4)*	0	0	0	x(-4)·x(0)*	x(-3)·x(-1)*
-3	x(-3)·x(-3)*	x(-2)·x(-4)*	0	0	0	0	0	x(-4)·x(-2)*
-4	x(-4)·x(-4)*	0	0	0	0	0	0	0

Figure 11. $A(\mu, n)$ for $N = 8$.

B. CONJUGATE SYMMETRY ADVANTAGE

Analysis shows that $A(\mu,-n)$ is equal to the conjugate of $A(\mu,n)$. That is, for $x_1 = A + jB$ and $x_2 = C + jD$;

$$x_1 \cdot x_2^* = (A + jB)(C - jD) = (AC + BD) + j(BC - AD)$$
$$= \left[(AC + BD) - j(BC - AD)\right]^* = \left[(A - jB)(C + jD)\right]^*$$
$$= (x_2 \cdot x_1^*)^*$$

or

$$x_1 \cdot x_2^* = (x_2 \cdot x_1^*)^* \tag{3.1}$$

So

$$A(\mu,n) = \left[A(\mu,-n)\right]^*$$

for all $n \neq 0$. Also, note that the kernel function is strictly symmetric about n

$$\phi(\mu,\ell,n) = \frac{1}{\sqrt{4\pi n^2/\sigma}} e^{-\frac{(\mu-\ell)^2}{4n^2/\sigma}}$$

so

$$\phi(\mu,\ell,n) = \phi(\mu,\ell,-n)$$

Therefore

$$\phi(\mu,\ell,n)A(\mu,n) = \left[\phi(\mu,\ell,-n)A(\mu,-n)\right]^* \tag{3.2}$$

Finally

$$\sum_{i=x}^{y}\left(f(i)\right)^* = \left[\sum_{i=x}^{y}\left(f(i)\right)\right]^* \tag{3.3}$$

and

$$S(\ell,-n) = \left[S(\ell,n)\right]^* \tag{3.4}$$

and

$$S'(\ell,n) = \left[S'(\ell,N-n)\right]^* \quad \text{for } N/2+1 \le n \le N-1 \tag{3.5}$$

Since the array for $S'(\ell,n)$ is symmetrical in this way, only the values of $S'(\ell,n)$ for $0 \le n \le N/2 - 1$ need be computed. Then, the conjugates of the function for $1 \le n \le N/2 - 1$ can be reverse ordered and used as the results of the function for $N/2 + 1 \le n \le N - 1$. This reduces the calculations for $S'(\ell,n)$ by another factor of two (for large values of N). The only cost in computing time is the need to change the sign of the imaginary portion of the real number and copy the appropriate values into the array. Figure 12 shows how $A(\mu,n)$ maps to $\left[A(\mu,-n)\right]^*$.

$\mu \setminus n$	0	1	2	3	-4	-3	-2	-1
3	x(3)·x(3)*							
2	x(2)·x(2)*	A						A*
1	x(1)·x(1)*	B	G				G*	B*
0	x(0)·x(0)*	C	H	K		K*	H*	C*
-1	x(-1)·x(-1)*	D	I	L		L*	I*	D*
-2	x(-2)·x(-2)*	E	J				J*	E*
-3	x(-3)·x(-3)*	F						F*
-4	x(-4)·x(-4)*							

Figure 12. $A(\mu,n)$ for $N = 8$.

The next optimization takes further advantage of the symmetric nature of the CWD and will increase the speed of computation with no deleterious effects. Note that (3.5) establishes the conjugate symmetricity of the function that is input into the FFT:

$$S'(\ell,n)=\left[S^{'}(\ell,N-n)\right]^{*}$$

for $N/2+1\le n\le N-1$. Taking a closer look at (2.6):

$$CWD_x(\ell,\frac{\pi k}{N})=2\sum_{n=0}^{N-1}S^{'}(\ell,n)e^{-j2\pi k\frac{n}{N}}$$

the summation representing the Fourier transform of the input array may be manipulated as follows. The summation can be broken into two parts.

$$\sum_{n=0}^{N-1}S^{'}(\ell,n)e^{-j2\pi k\frac{n}{N}}=\sum_{n=0}^{N/2-1}S^{'}(\ell,n)e^{-j2\pi k\frac{n}{N}}+\sum_{n=N/2}^{N-1}S^{'}(\ell,n)e^{-j2\pi k\frac{n}{N}}$$

Taking further advantage of 3.5, the second term ends up being the conjugate of the first term, minus the result of the function for $n=0$.

$$\sum_{n=N/2}^{N-1}S^{'}(\ell,n)e^{-j2\pi k\frac{n}{N}}=S^{'}(\ell,N/2)e^{-j2\pi k\frac{N/2}{N}}+\sum_{n=N/2+1}^{N-1}S^{'}(\ell,n)e^{-j2\pi k\frac{n}{N}}$$

$$=0+\sum_{n=N/2+1}^{N-1}S^{'}(\ell,N-n)^{*}e^{-j2\pi k\frac{n}{N}}$$

Using the substitution $n'=N-n$

$$\sum_{n=N/2+1}^{N-1}S^{'}(\ell,N-n)^{*}e^{-j2\pi k\frac{n}{N}}=\sum_{n'=N/2-1}^{1}S^{'}(\ell,n')^{*}e^{-j2\pi k\frac{N-n'}{N}}$$

$$=\sum_{n=1}^{N/2-1}S^{'}(\ell,n)^{*}e^{-j(-2\pi k\frac{n}{N}+2\pi k)}=\sum_{n=1}^{N/2-1}S^{'}(\ell,n)^{*}e^{j2\pi k\frac{n}{N}}$$

or

$$\sum_{n=N/2}^{N-1}S^{'}(\ell,n)e^{-j2\pi k\frac{n}{N}}=\left[\sum_{n=1}^{N/2-1}S^{'}(\ell,n)e^{-j2\pi k\frac{n}{N}}\right]^{*}$$

Since the summing of a complex number with its conjugate will equal double the real portion of the number ($A + A^* = 2\,\mathrm{Re}\{A\}$) and $S'(\ell, 0)$ will always be a real number, the total summation then becomes:

$$\sum_{n=0}^{N-1} S'(\ell, n) e^{-j2\pi k \frac{n}{N}} = 2\,\mathrm{Re}\left\{ \sum_{n=0}^{N/2-1} S'(\ell, n) e^{-j2\pi k \frac{n}{N}} \right\} - S'(\ell, 0) \tag{3.6}$$

This result can be used in the following way. A new function $S^H(\ell, n)$ can be defined as:

$$S^H(\ell, n) = \begin{cases} \frac{1}{2} S'(\ell, n) & n = 0 \\ S'(\ell, n) & 1 \le n \le N/2 - 1 \\ 0 & \textit{elsewhere} \end{cases} \tag{3.7}$$

The CWD then becomes:

$$CWD_x(\ell, \frac{\pi k}{N}) = 4 \sum_{n=0}^{N-1} S^H(\ell, n) e^{-j2\pi k \frac{n}{N}} \tag{3.8}$$

Using this equation, the exact same real results are produced, but only the left half of the input array need be computed. The imaginary part of the original equation always, by definition, summed to zero. In the new equation, the imaginary part is simply discarded. For $N/2 \le n \le N - 1$, all input values into the FFT are zero. This manipulation was verified using MATLAB and the programming language C. The algorithm was modified, as described above, and the exact same results were obtained.

The next section discusses the “cut and slice” optimization, which eliminates the computation of near zero terms stemming from very small values in the kernel function.

C. CUT AND SLICE

All optimizations made up to this point have had no affect on the final result of the computation. This next optimization changes the final result of the computation. An analysis of the changes affected by this optimization is provided in the next chapter.

Another way to vastly improve the speed with which the distribution can be computed is to take advantage of the properties of the kernel. Figures 13 through 15 show that much of the kernel function appears to be near zero.

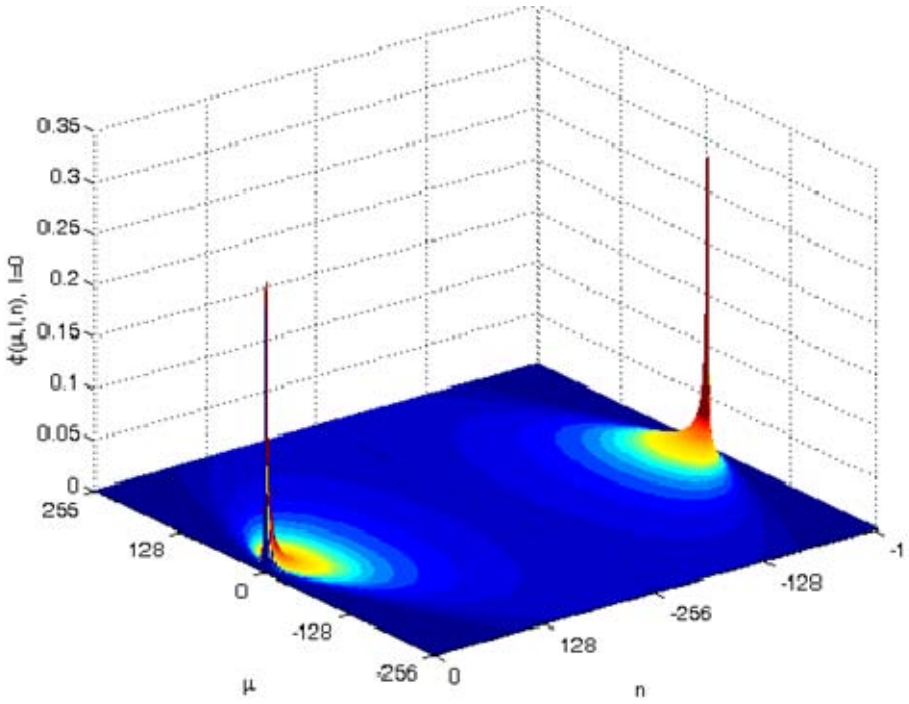


Figure 13. Choi-Williams kernel function $\ell = 0$, $N = 512$.

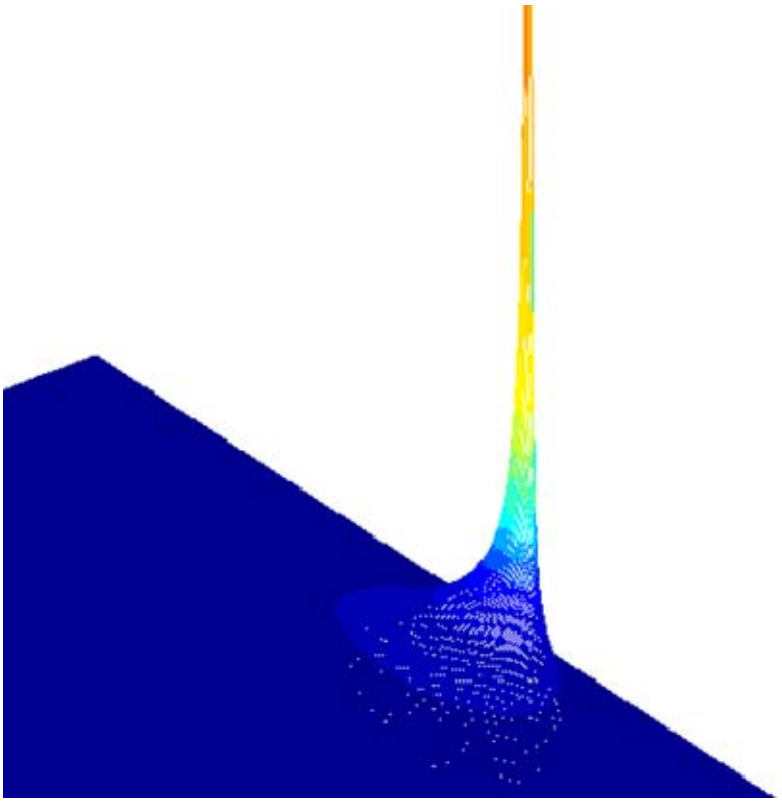


Figure 14. Choi-Williams kernel function $\ell = 0$, $N = 512$.

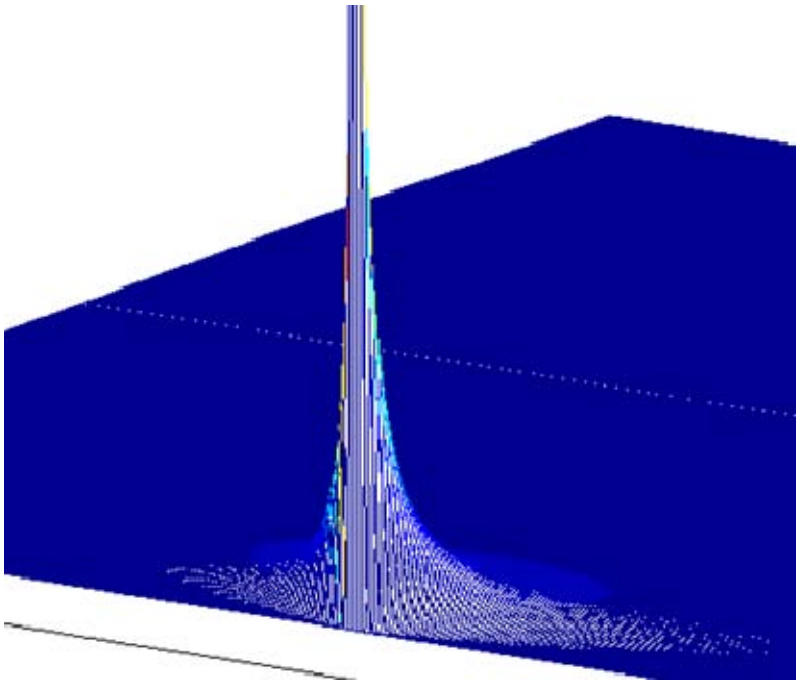


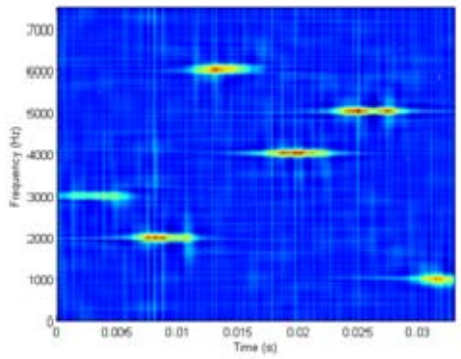
Figure 15. Choi-Williams kernel function $\ell = 0$, $N = 512$.

Indeed, looking at the kernel function again,

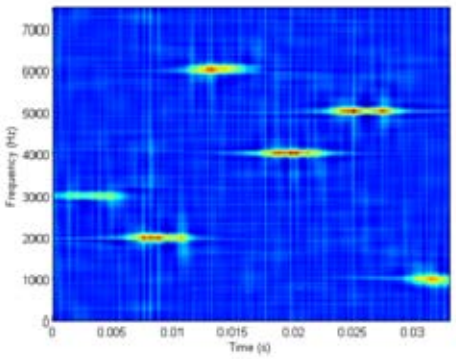
$$\phi(\mu,\ell,n)=\frac{1}{\sqrt{4\pi n^2/\sigma}}e^{-\frac{(\mu-\ell)^2}{4n^2/\sigma}}$$

it can be seen that the further away from $(\mu-\ell)=0$ and $n=0$, the closer the kernel function gets to zero. Since any number multiplied by near zero will have little contribution to a summing operation, it can be reasoned that eliminating the calculations that will produce near zero results will reduce the computation time while having little affect on the final result. An analysis was conducted to subjectively measure the degradation in results of reducing near zero terms and determine a threshold that does not degrade the results noticeably. A subjective analysis was used instead of a deterministic one because the ability of this signal processing technique to provide a clear picture of the signal cannot be determined by any function.

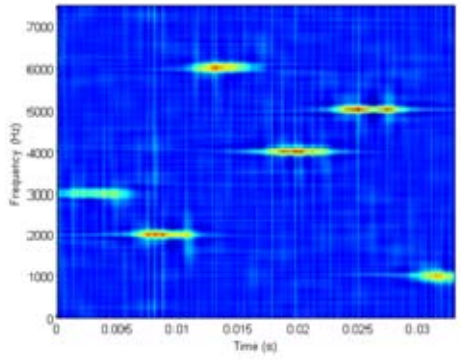
To conduct this analysis, the weighting kernel function was first "cut." The variable "cut" was defined as the amount of columns surrounding the peak of the kernel function that were used in the computation. For example, a "cut" of 256 would include the entire 512 columns. A cut of 128 would include the first 128 columns and the last 128 columns. The variable "slice" was used to eliminate rows above and below the peak of the kernel function. A "slice" of 64 would include the row with the peak of the weighting function and the 63 rows both above and below the peak of the kernel function. First, the kernel was cut to find a threshold where the picture began to degrade. As can be seen in Figure 16, the picture is degraded noticeably for cut < 32. Note that, cut = 256, slice = 512, is identical to the original computation.



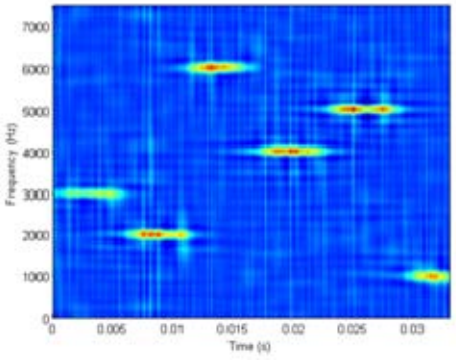
cut = 256, slice = 512;



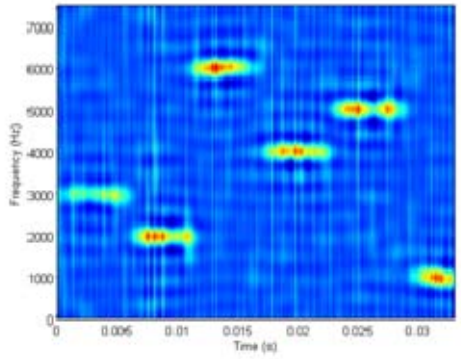
cut = 128; slice = 512



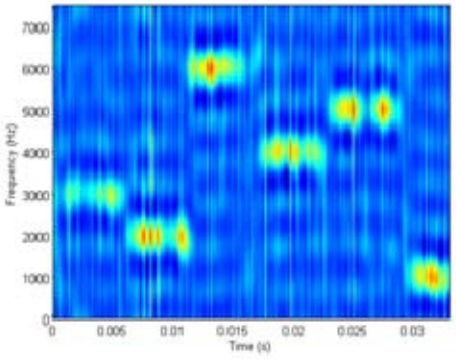
cut = 64, slice = 512;



cut = 32; slice = 512



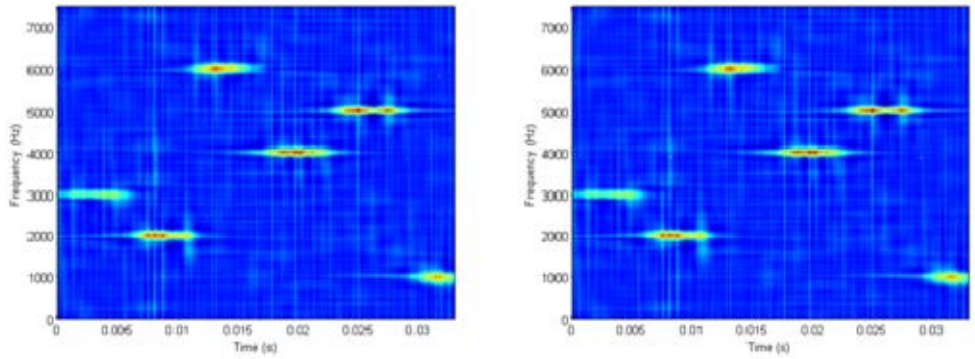
cut = 16, slice = 512;



cut = 8; slice = 512

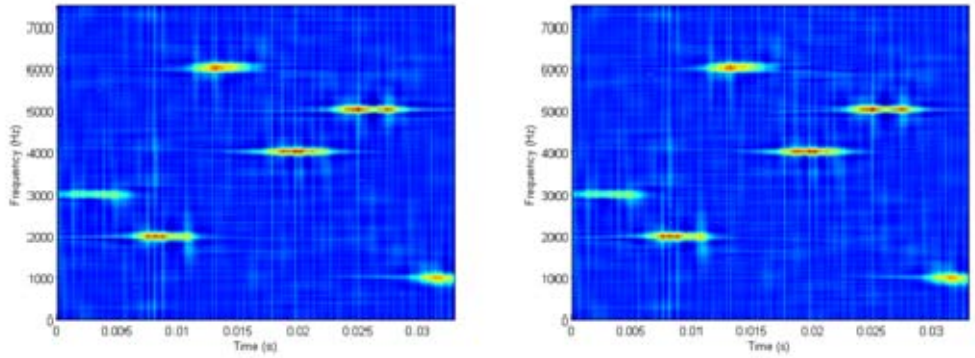
Figure 16. Evaluation of the Cut method on the Costas frequency hopping signal.

Figure 17 shows a similar analysis for “slice”. The subjective threshold for degradation was determined visually to be slice = 32.



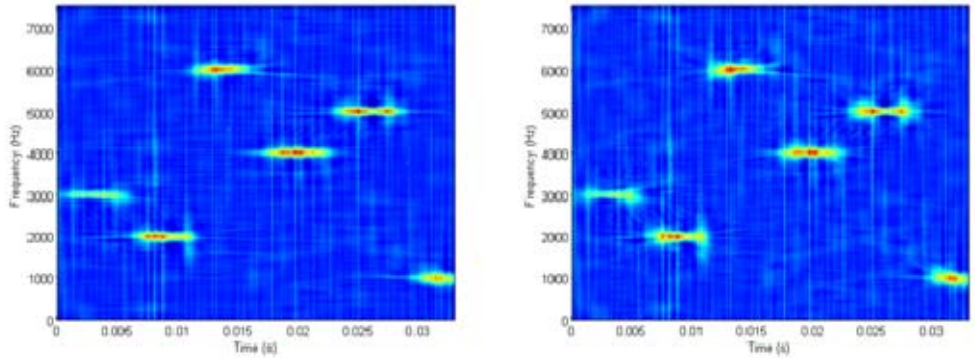
cut = 256 slice = 512

cut = 256 slice = 256



cut = 256 slice = 128

cut = 256 slice = 64

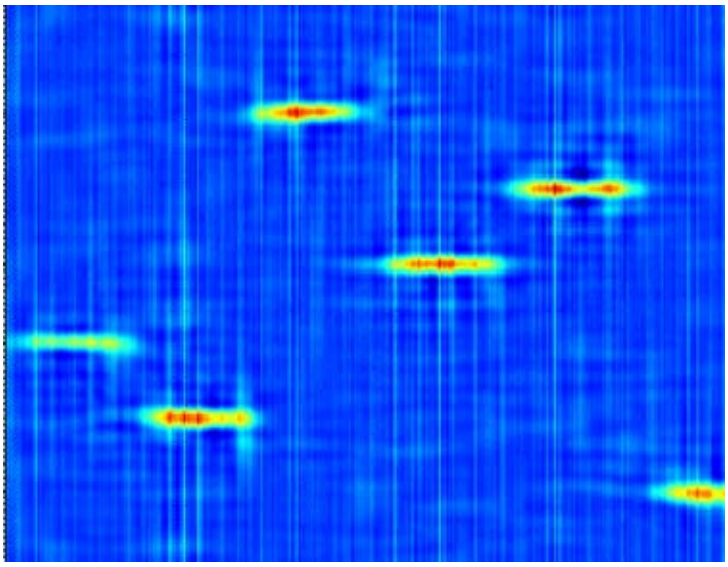


cut = 256 slice = 32

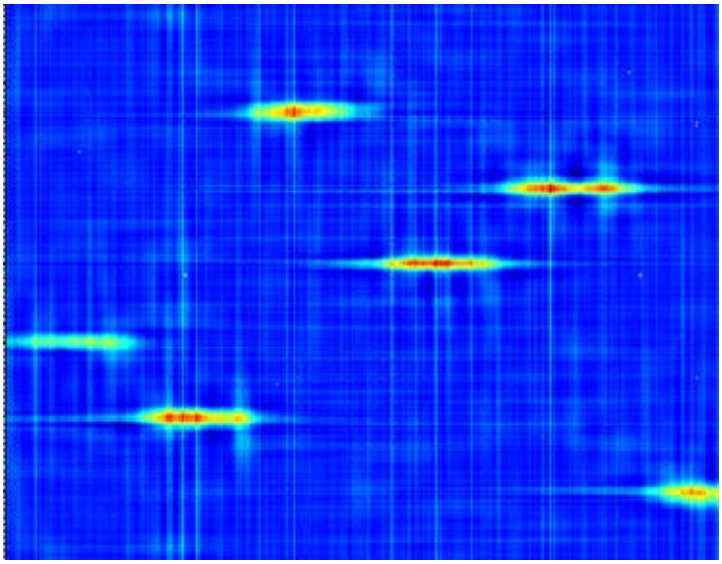
cut = 256 slice = 16

Figure 17. Evaluation of the Slice method on the Costas frequency hopping signal.

Next, cut and slice were conducted at the two thresholds at the same time to verify that the effects of the two optimizations do not have a multiplicative affect. Figure 18 shows that reducing the computations to thresholds of 32 rows and 32 columns surrounding the peak of the kernel function produces an image that is sufficiently similar to the full version as to allow the picture of the signal to still be readable.



Cut = 32 Slice = 32



Cut = 256 Slice = 512 (Full Version)

Figure 18. Comparison of reduced computation and full computation results.

Figure 18 shows the kernel function that has been cut and sliced. Since all rows and columns more than 32 away from the peak of the function are zero, and therefore produce a result of zero when multiplied by $A(\mu, n)$, these products are simply not computed. In Figure 19, cut is the shorter axis, slice is the long axis. Slice = 32 means 32 to either side of 0.

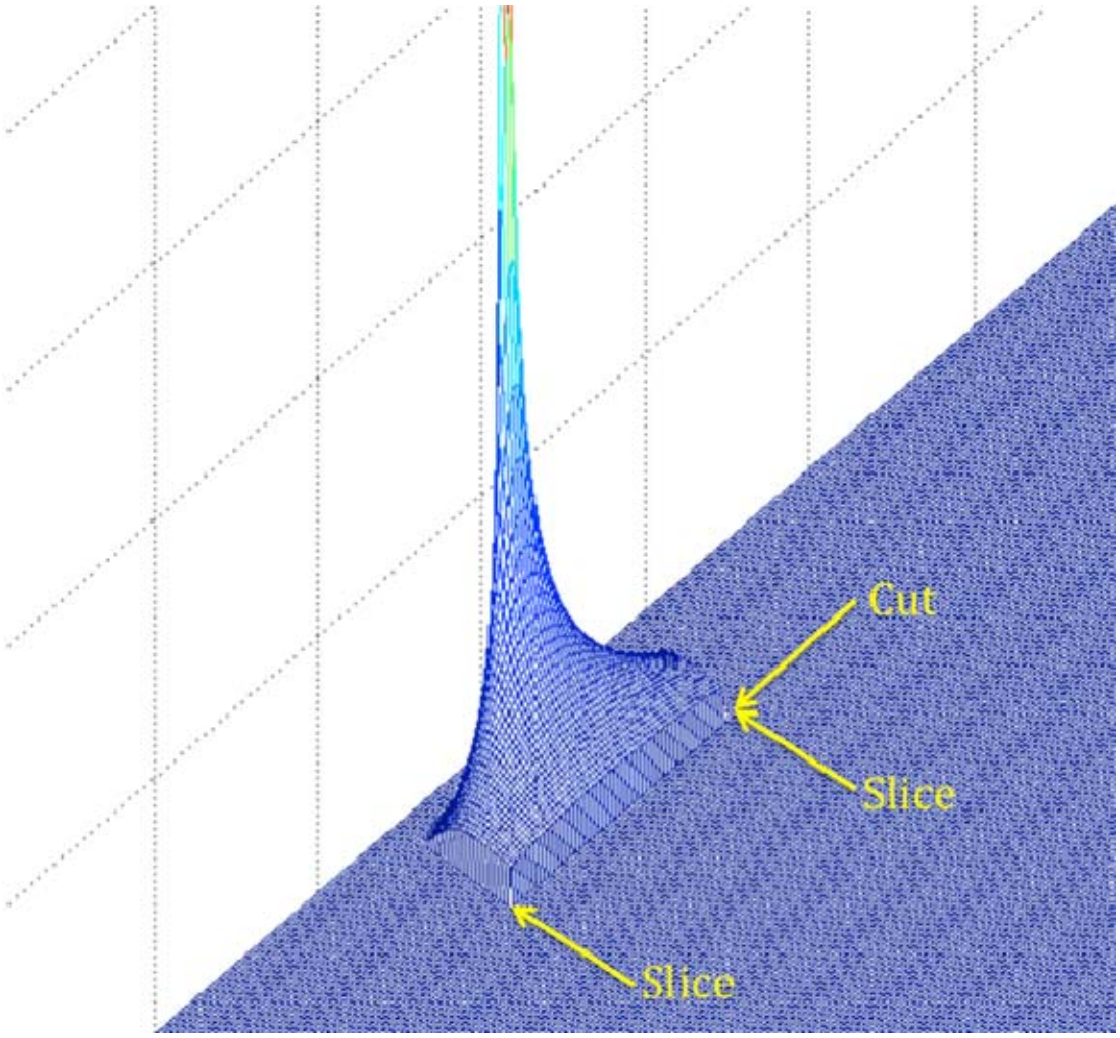


Figure 19. The kernel function with Cut = 32, Slice = 32.

The next section shows how the FFT algorithm can be modified to take advantage of all the zeroed values within the kernel function.

D. REDUCED FFT

The next optimization takes advantage of all the zeros produced by the previous optimizations. This optimization increases the speed of the FFT computation by $9/5$. Since, without this optimization, the FFT accounts for roughly half of the total computation time for the CWD, this is significant improvement.

Figure 20 shows a common way to implement an FFT using $\log_2 n$ layers of BFMs to implement an n point FFT. Figure 20 shows an eight point FFT implemented with three layers of BFMs.

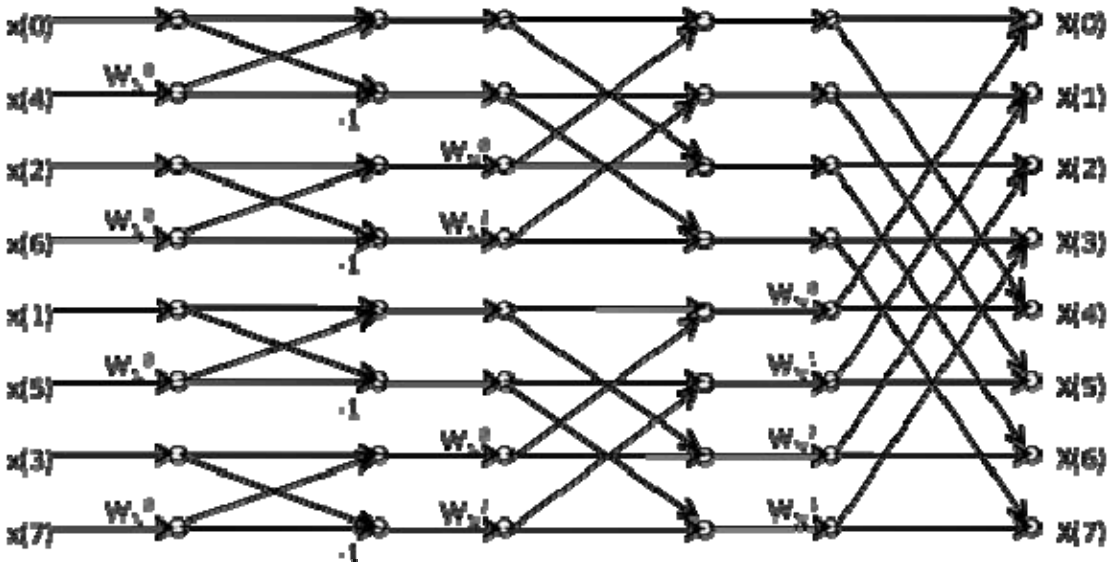


Figure 20. 8-point FFT structure [From 13].

In Figure 20, arrows joining at a circle indicate that two values are summed. The various weighting terms shown represent the twiddle factors that are multiplied where $W_N^k = \exp\left(-j\,2\pi k/N\right)$. The negative ones indicate that the arrow is multiplied by a negative one. Figure 21 shows how the algorithm is altered when all but the first two inputs (x(0) and x(1)) are made equal to zero.

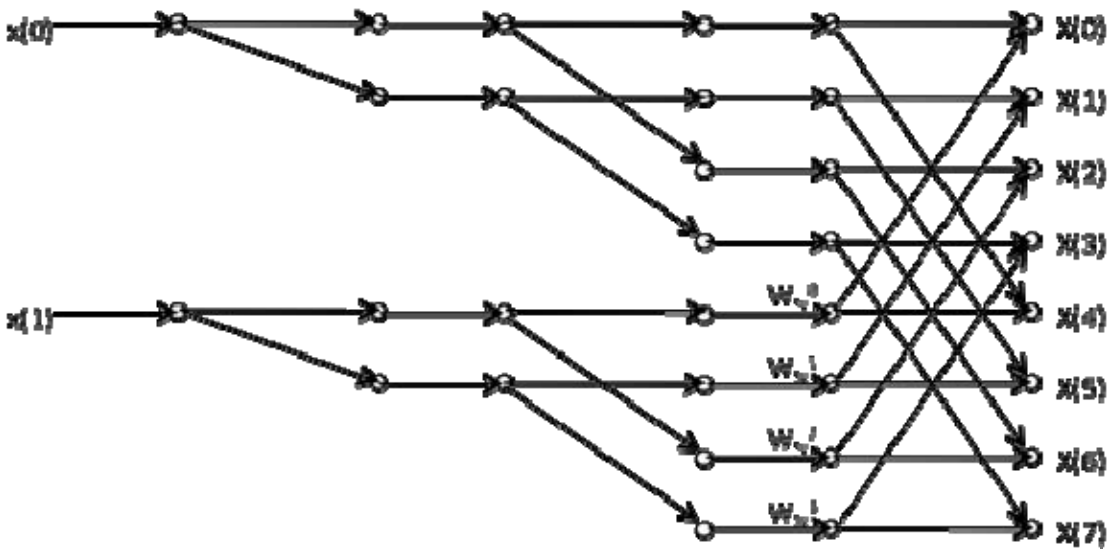


Figure 21. Modified FFT [modified From 13].

Notice that the first two layers of BFMs can be completely skipped and the inputs can be fed directly into the third layer. This same affect works on FFTs of this type for all sizes with the following conditions and effects:

1. For any FFT calculated for n inputs where $\log_2 n$ is a whole number.
2. If every input following the first m values is equal to 0.
3. And $\log_2 m$ is a whole number.
4. Then the m inputs can be fed directly into the $\left(\log_2 \frac{n}{m} + 1\right) th$ BFM layer of the n point FFT.
5. Each input will be fed into $\frac{n}{m}$ adjacent inputs.
6. And the order of the inputs is in the order of the values represented by the reversed order bit representations of the values zero through $m-1$ represented in $\log_2 m$ bits.

For the example above:

1. $n = 8 \quad \log_2 n = 3$
2. Every input past the first two inputs is zero.
3. $m = 2 \quad \log_2 m = 1$
4. The first two values are fed directly into the $\log_2 \frac{8}{2} + 1 = 3$rd layer of the 8-point FFT.
5. Each input is fed into $\frac{8}{2} = 4$ adjacent inputs.
6. The values zero through $2 - 1 = 1$ are represented in $\log_2 2 = 1$ bits. A bit reversal is conducted (trivial in this case), and the order of input is established by these reverse bit values (again in this case no reordering is required).

For the purposes of implementing the above optimization for our 512 input FFT with a "cut" of 32, limiting the input to only 32 values:

1. $n = 512 \qquad \log_2 n = 9$
2. Every input past the first 32 inputs is zero.
3. $m = 32 \qquad \log_2 m = 5$
4. The first 32 values are fed directly into the $\log_2 \frac{512}{32} - 1 = 5th$ layer of a 512 point FFT.
5. Each input is fed into $\frac{512}{32} = 16$ adjacent inputs.
6. The values zero though 31 are represented in 5 bits. A bit reversal is conducted and the order of inputs is established by these reverse bit values (see Table 2).

0-31	Reversed bit	Order of inputs
0	00000	0
1	10000	16
2	01000	8
3	11000	24
4	00100	4
…	…	…
31	11111	31

Table 2. Order of inputs.

The next section shows an optimization that might be beneficial if the CWD were computed using a reconfigurable computer.

E. AN OPTIMIZATION NOT REALIZED

The next optimization was not realized, due to the fact that implementation using C code would probably provide a negligible, if any, increase in speed of computation. It is, however, an interesting optimization in that it uses powers of two, instead of the exponential function, making it a potentially powerful optimization to use with reconfigurable computers. Also, this section takes a closer look at the mysterious parameter known only as “sigma”.

From (2.5) we start with $\phi(\mu, \ell, n)$:

$$\phi(\mu, \ell, n) = \frac{1}{\sqrt{4\pi n^2 / \sigma}} e^{-\frac{(\mu-\ell)^2}{4n^2/\sigma}}$$

Using the identity

$$2^{\log_2(x)} = x$$

the following simplification can be made

$$\sqrt{\frac{\sigma}{4\pi n^2}}e^{-\frac{(\mu-\ell)^2\sigma}{4n^2}} = \frac{1}{n}\sqrt{\frac{\sigma}{4\pi}}2^{\log_2\left(e^{-\frac{(\mu-\ell)^2\sigma}{4n^2}}\right)}$$

then using the identity

$$\log_x y^z = z\log_x y$$

the following simplification can be made

$$\frac{1}{n}\sqrt{\frac{\sigma}{4\pi}}2^{\log_2\left(e^{-\frac{(\mu-\ell)^2\sigma}{4n^2}}\right)} = \frac{1}{n}\sqrt{\frac{\sigma}{4\pi}}2^{-\frac{(\mu-\ell)^2\sigma}{4n^2}\log_2(e)}$$

then using the identity

$$\log_x\left(y\right) = \frac{\log_z\left(y\right)}{\log_z\left(x\right)}$$

the same equation can be represented as follows:

$$\frac{1}{n}\sqrt{\frac{\sigma}{4\pi}}2^{-\frac{(\mu-\ell)^2\sigma}{4n^2}\log_2(e)} = \frac{1}{n}\sqrt{\frac{\sigma}{4\pi}}2^{-\frac{(\mu-\ell)^2\sigma}{4n^2}\frac{\ln(e)}{\ln(2)}}$$

or finally

$$\phi(\mu,\ell,n) = \frac{1}{\sqrt{4\pi n^2/\sigma}}e^{-\frac{(\mu-\ell)^2}{4n^2/\sigma}} = \frac{1}{n}\sqrt{\frac{\sigma}{4\pi}}2^{-\frac{(\mu-\ell)^2}{n^2}\frac{\sigma}{4\ln(2)}}$$

Before proceeding further, let us take a closer look at σ. Sigma changes the distribution of the kernel function. Up until this point, all computations in this thesis have been done with $\sigma = 1$. Figures 22 through 29 represent the kernel function for various values of σ and its affect upon the processing result of the CWD, using our running example of a Costas frequency hopping signal with $\mathrm{SNR} = 0\,\mathrm{dB}$.

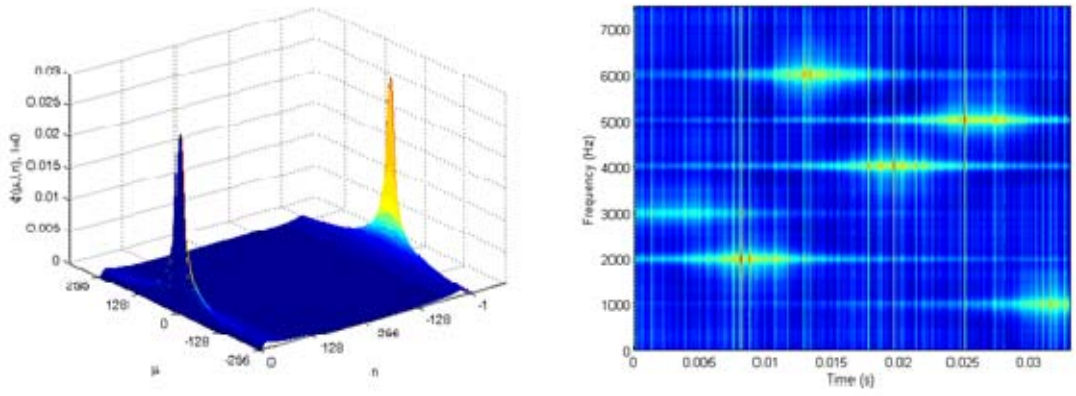


Figure 22. $\sigma = 0.01$.

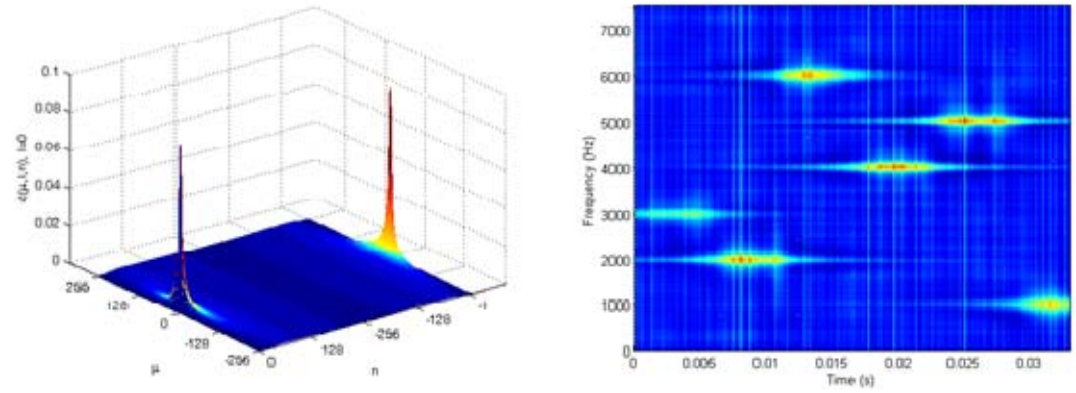


Figure 23. $\sigma = 0.1$.

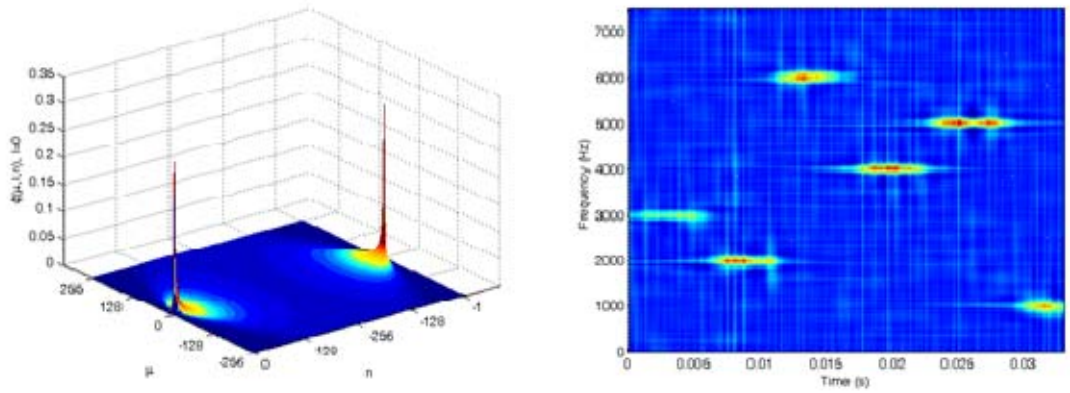


Figure 24. $\sigma = 1$.

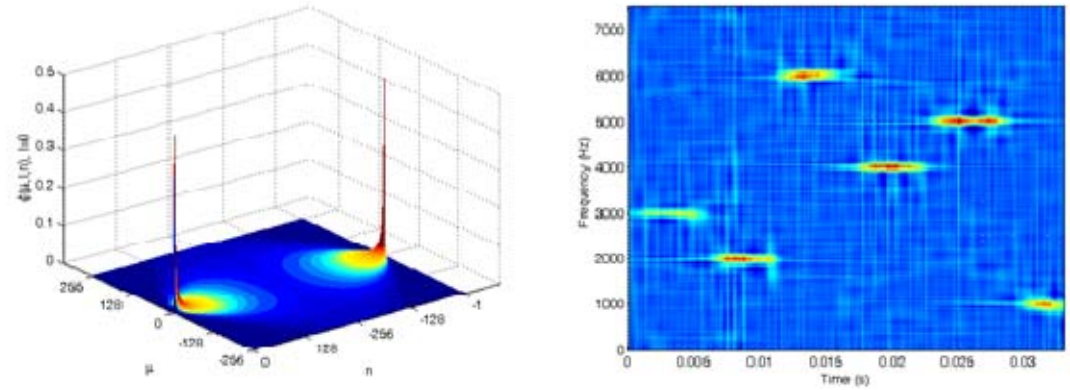


Figure 25. $\sigma = 2.77258872224 = 4\ln(2)$.

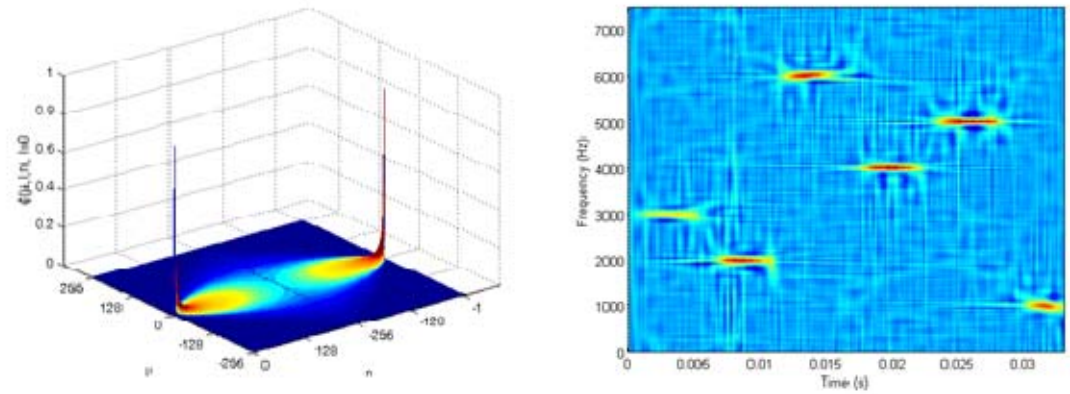


Figure 26. $\sigma = 10$.

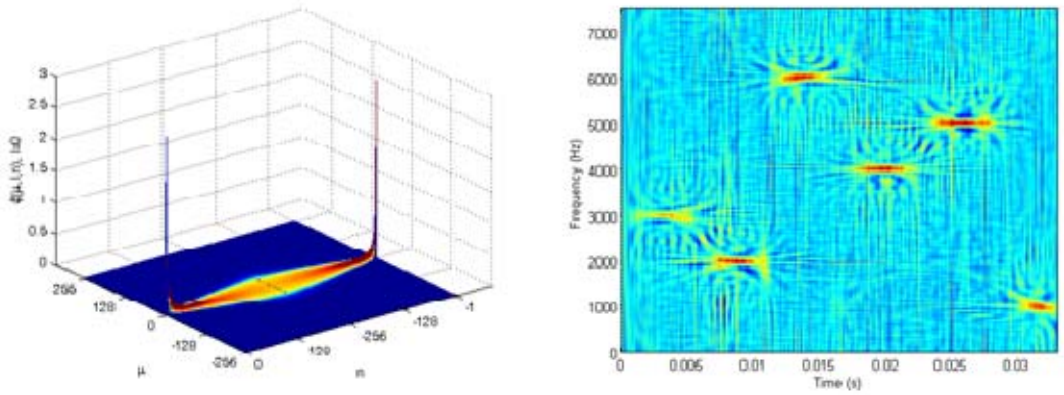


Figure 27. $\sigma = 100$.

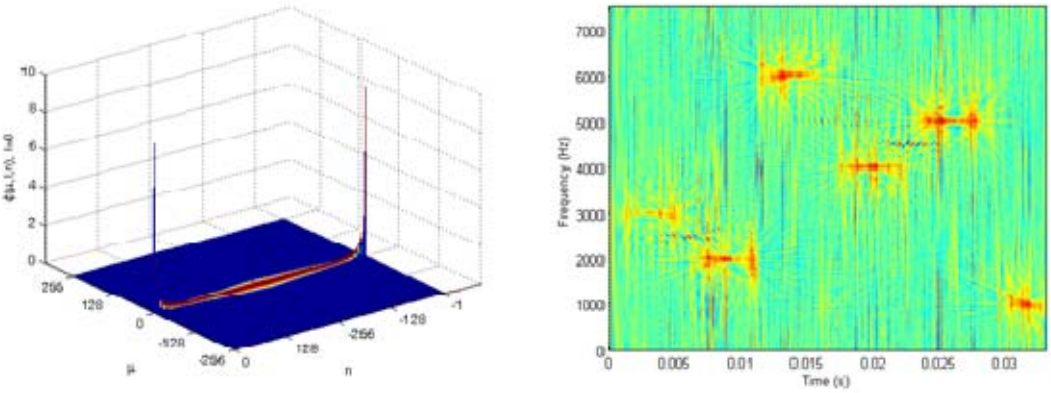


Figure 28. $\sigma = 1000$.

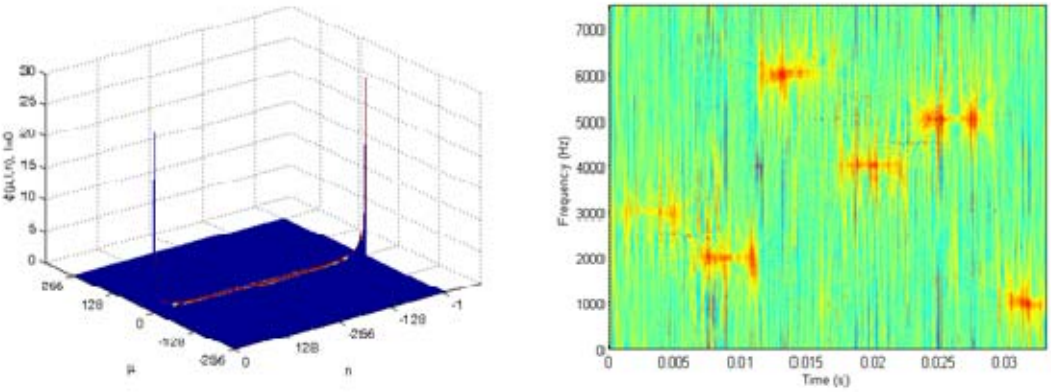


Figure 29. $\sigma = 10,000$.

Notice that in Figure 24 for $\sigma = 2.77258872224 = 4\ln(2)$ (a constant), the signal representation is good and the kernel function becomes:

$$\phi(\mu,\ell,n) = \frac{1}{n}\sqrt{\frac{\sigma}{4\pi}}\,2^{-\frac{(\mu-\ell)^2}{n^2}\frac{\sigma}{4\ln(2)}} = \frac{0.220635600153}{n}\,2^{-\left(\frac{\mu-\ell}{n}\right)^2} \qquad (3.9)$$

Note that (3.9) provides the mathematically same results as (2.5) and Figure 25 with $\sigma = 2.77258872224$, using the original kernel function:

$$\phi(\mu,\ell,n) = \frac{1}{\sqrt{4\pi n^2/\sigma}}\,e^{-\frac{(\mu-\ell)^2}{4n^2/\sigma}}$$

Figure 30 shows that if we round all values of the exponent term, $\left((\mu-\ell)/n\right)^2$, to the nearest integer, we get the following kernel function and results for our running example of the Costas signal. Figure 31 shows a close-up of the modified kernel function with its exponent rounded. Notice that the plot of the rounded kernel function is not as smooth as the unrounded kernel function, but that it still retains the same overall shape and characteristics.

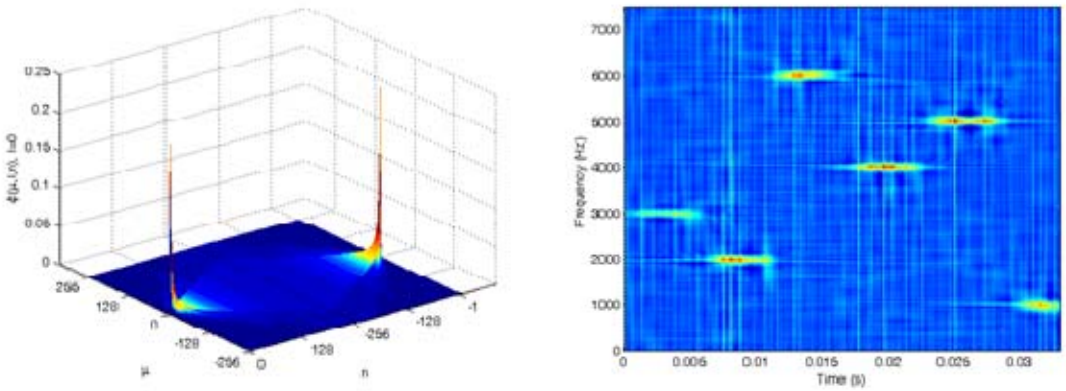


Figure 30. Rounded kernel function, $\sigma = 4\ln(2)$.

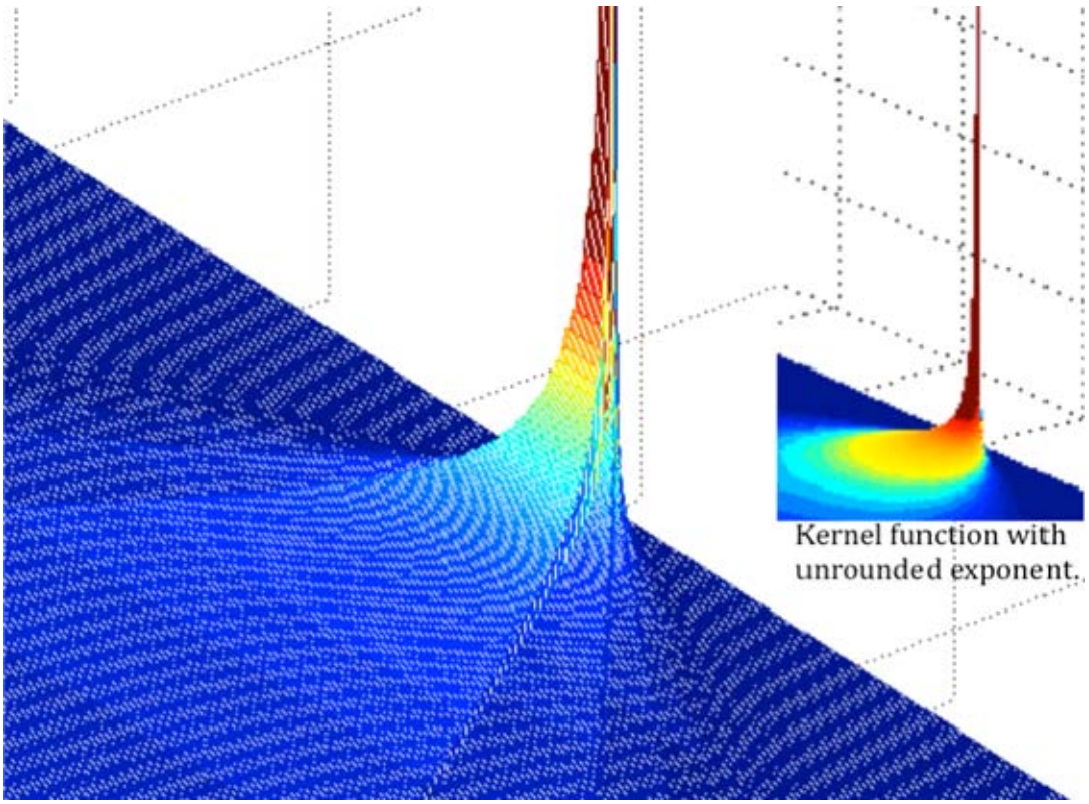


Figure 31. Comparison of the rounded kernel function, $\sigma = 4\ln(2)$, with the unrounded kernel function.

Although the effect of rounding the kernel function to the nearest power or two is noticeable, the effects upon the output distribution are negligible. Figure 32 compares the two outputs generated for Figures 25 and 30. It is difficult to tell the difference between the two.

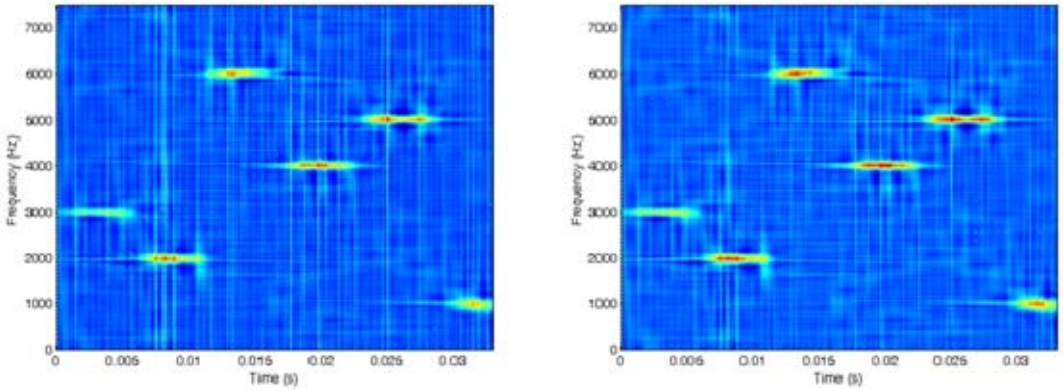


Figure 32. Comparing rounded to unrounded version, $\sigma = 4\ln(2)$.

If the new rounded version is used, for every $S'(\ell,n)$ computed, $S'(\ell+1,n)=S'(\ell,n)2^{\alpha}$ where α is some integer (equally applicable to $S^{H}(\ell+1,n)=S^{H}(\ell,n)2^{\alpha}$). If IEEE floating point is used to represent the number, then $A(\mu,n)$ need only be weighted once by multiplying it by a kernel function. Since the rest of the weighting kernel functions in a column are related by some power of two, the mantissa of the floating point representation will remain the same for all other weights. The other weights can be calculated by adding or subtracting to the eight-bit exponent of the floating point representation. Since there is no such single operation in the C programming language, to utilize this method would involve masking out the eight-bit exponent, modifying it, and then placing it back into the number. The benefits of this multiple instruction operation over simply doing the multiplications are questionable. However, if a reconfigurable computer is used to perform the operation, there is the potential for substantial savings in computation time.

This is the last optimization examined. The next chapter takes a closer look at the effects of the cut and slice optimization, detailed in Section C of this chapter, and shows the overall improvement in computation speed attained by utilizing the optimizations.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ERROR AND TIMING ANALYSIS

A. THE (NOT SO) DELETERIOUS EFFECTS OF CUT AND SLICE

In establishing the threshold for the “cut and slice” optimization, the author used a subjective approach to determine how small the kernel function could be cut and sliced without degrading the quality of the results. This section conducts an objective analysis to show the effect of the cut and slice operation. Also, this section shows the effects of the optimization on some common LPI signals to determine the effect of the optimization on a range of signals at various signal to noise (SNR) ratios.

Five common LPI signals are analyzed—each one at three different levels of SNR: signal only ($\mathrm{SNR} = \infty$ dB), $\mathrm{SNR} = 0$ dB and $\mathrm{SNR} = -6$ dB. Each of the 15 example signals were measured at cut = slice = 128, 64, 32, 16, and 8 (see cut and slice section in the previous chapter for explanation of levels) and compared to the results of the unaltered version of the CWD of the same signal.

$$Error = median\left[\frac{|CWD - CS|}{\overline{CWD}}\right] \tag{4.1}$$

To compare each computation result to the original version, the absolute value of the difference between the cut and slice (CS) version and the original computation are measured at each of the 512 x 512 points in the array, and then they are divided by the average value of all the points in the original array. Then the median is taken (4.1).

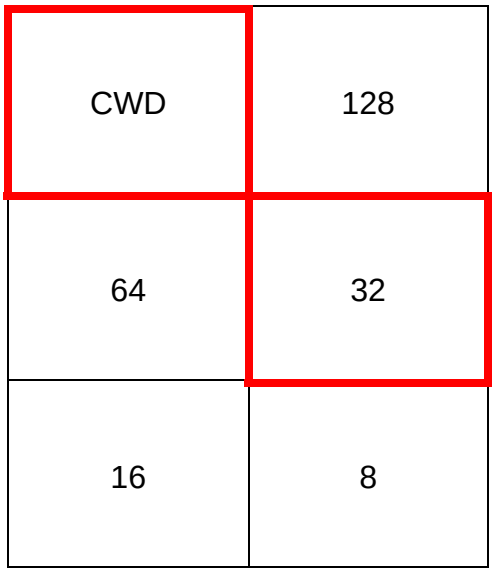


Figure 33. Figure Map

The results of the analysis are summarized in Figures 34 through 48 below. In each of the figures, there are six plots. The top left plot shows the result of the unmodified version of the Choi-Williams computation. The plot to its right is the result of the computation with cut = slice =128, which is the number of columns and rows surrounding the peak of the kernel function, $\phi(\mu = \ell, n = 0)$. The plot below the original is for cut = slice = 64, and to that one's right cut = slice = 32. The two plots on the bottom are for cut = slice = 16, and cut = slice = 8. Figure 33 above summarizes the layout. At the bottom of each set of time-frequency plots is a stem plot of the Error measured for each of the cut and slice computations compared to the original.

Note that cut = slice = 32 was the threshold established for optimizing the computation using the subjective approach and is the chosen threshold to use for the optimized version of the algorithm.

The five signals examined are: Frequency Modulated Carrier Wave (FMCW), Polyphase 1 (P1), Polytime 1 (PT1), Costas Frequency Hopping (Costas), and Frequency Shift Keying/Phase Shift Keying (FSK/PSK).

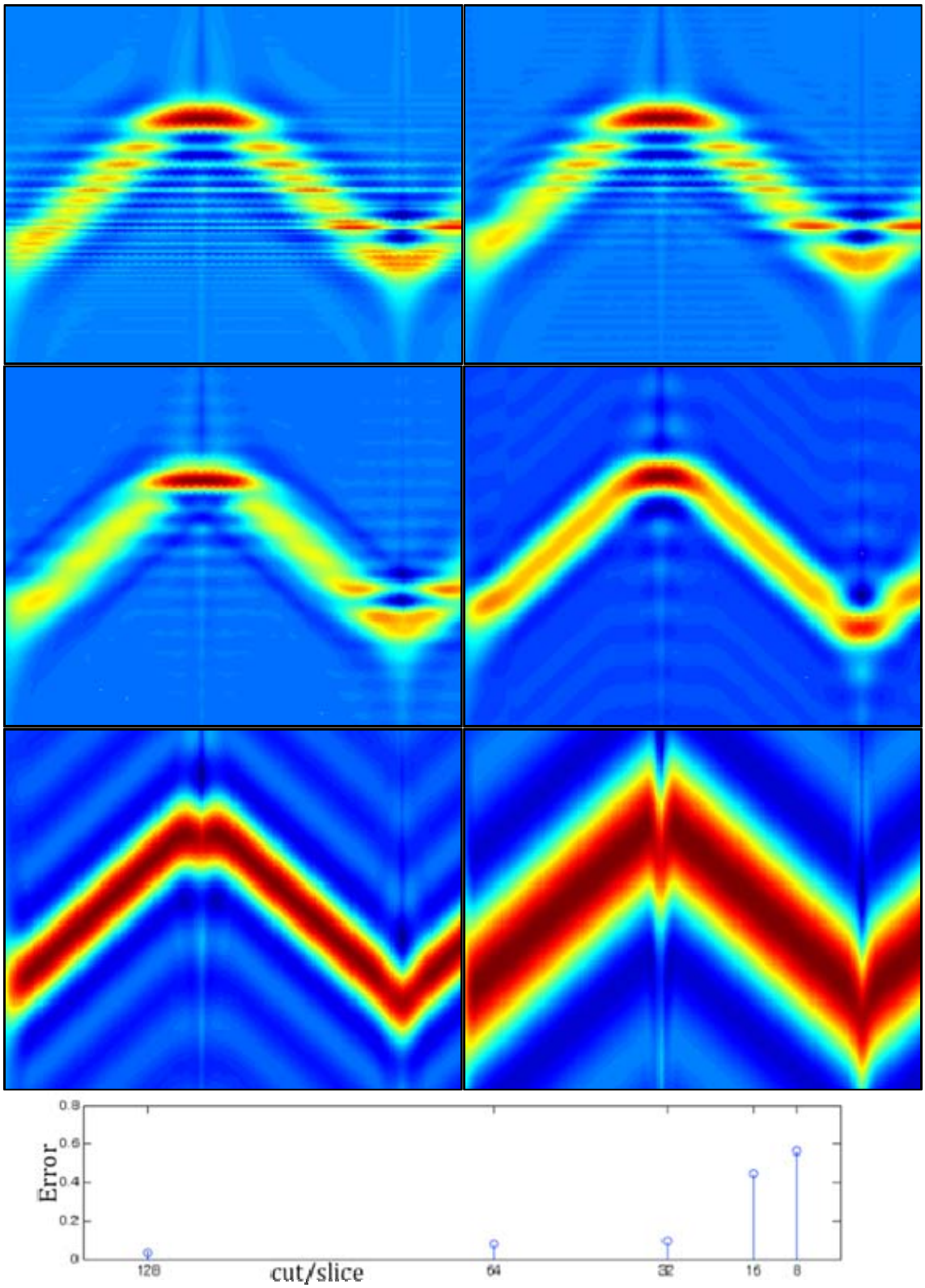


Figure 34. Cut and Slice results for FMCW, signal only, including a stem plot of the error when compared to the original (top left).

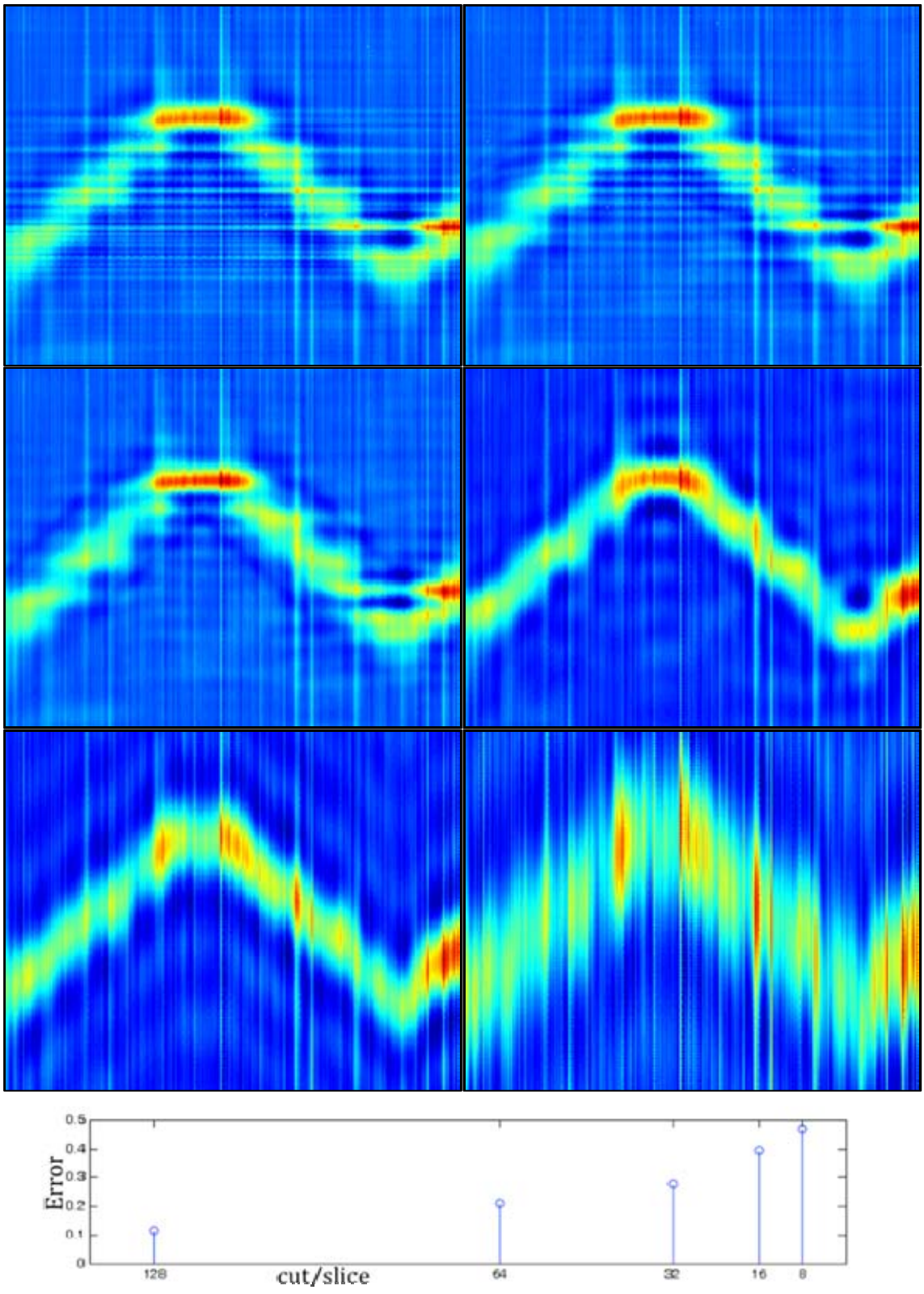


Figure 35. Cut and Slice results for FMCW, 0 dB, including a stem plot of the error when compared to the original (top left).

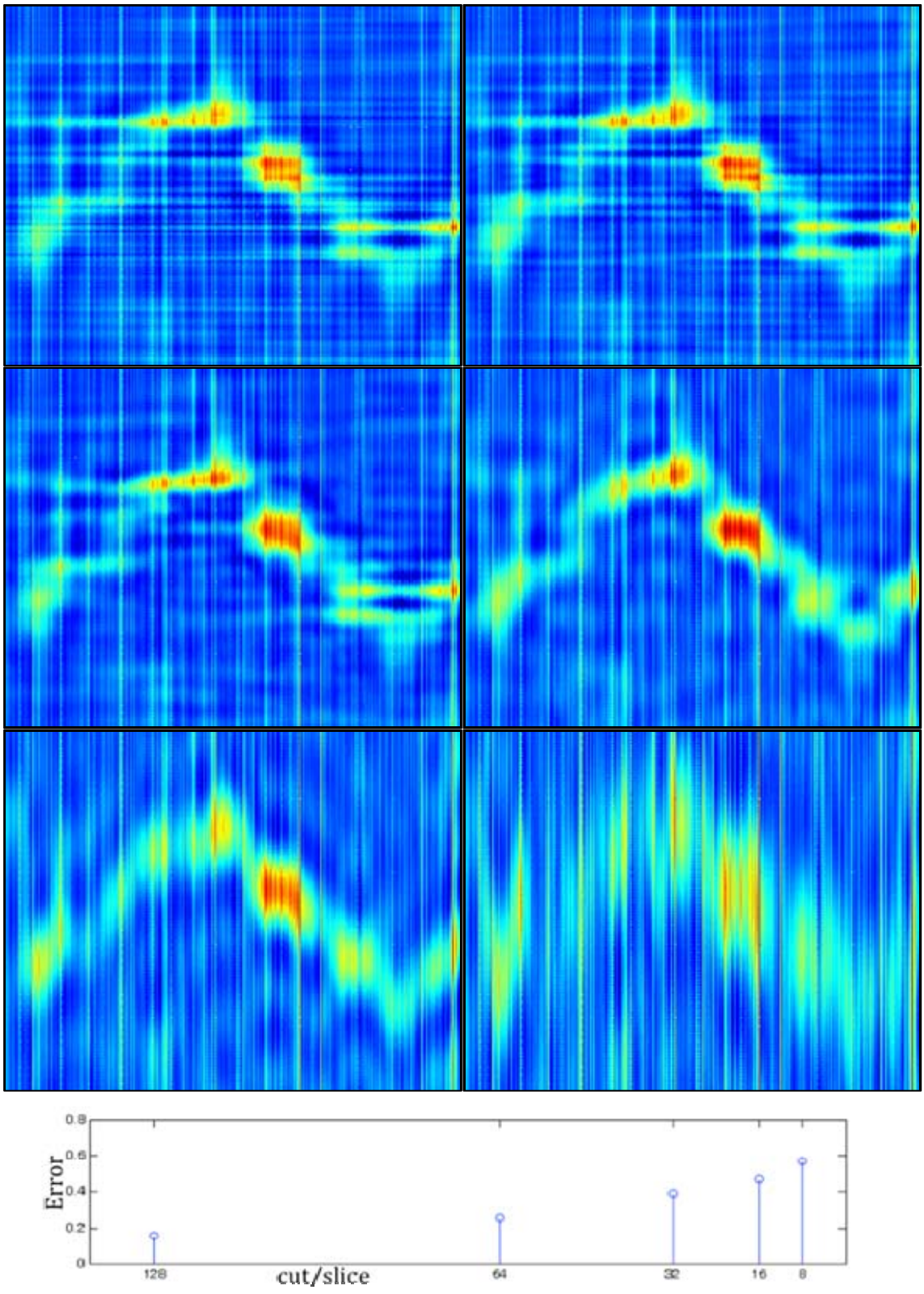


Figure 36. Cut and Slice results for FMCW, -6 dB, including a stem plot of the error when compared to the original (top left).

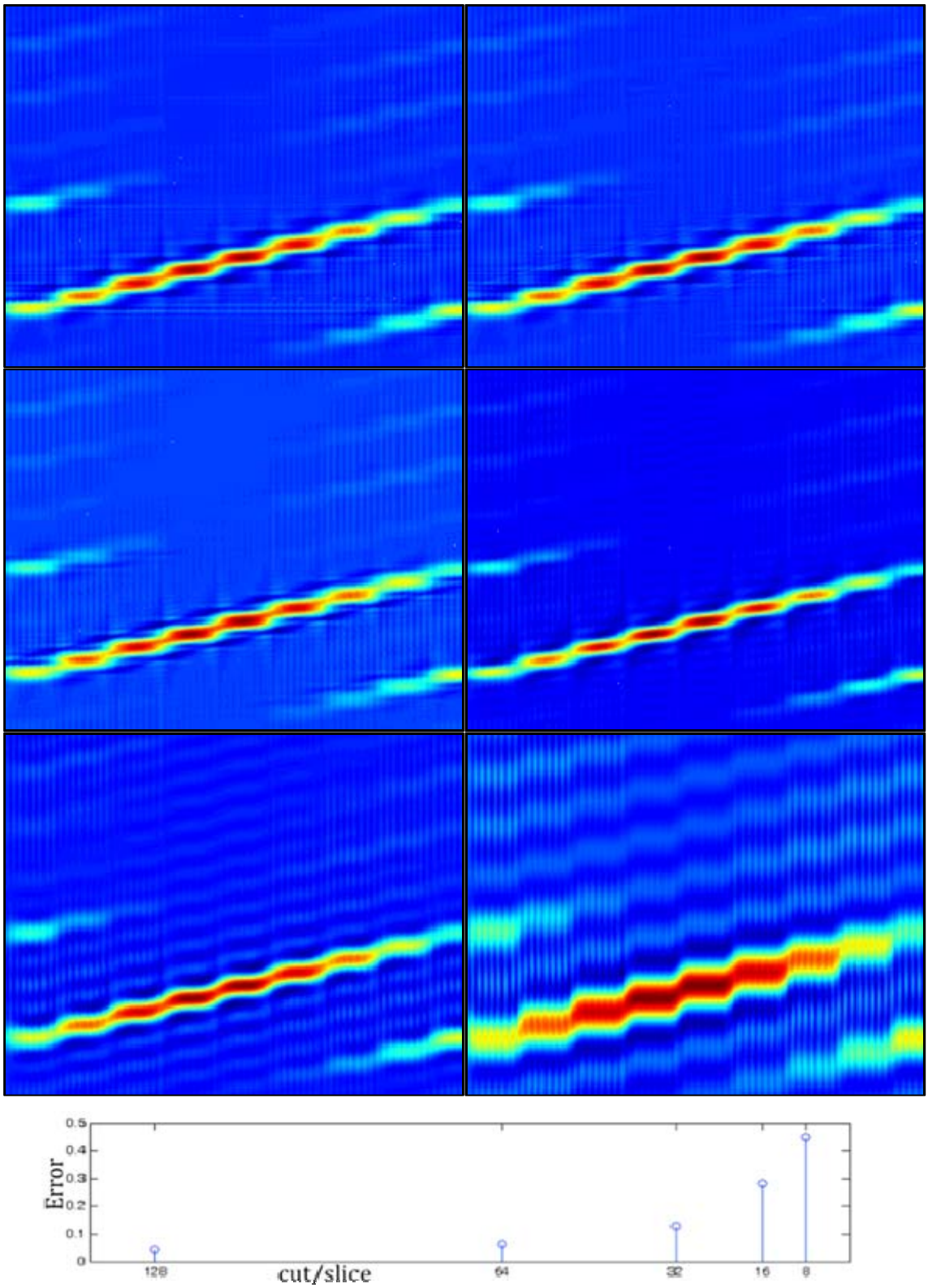


Figure 37. Cut and Slice results for P1, signal only, including a stem plot of the error when compared to the original (top left).

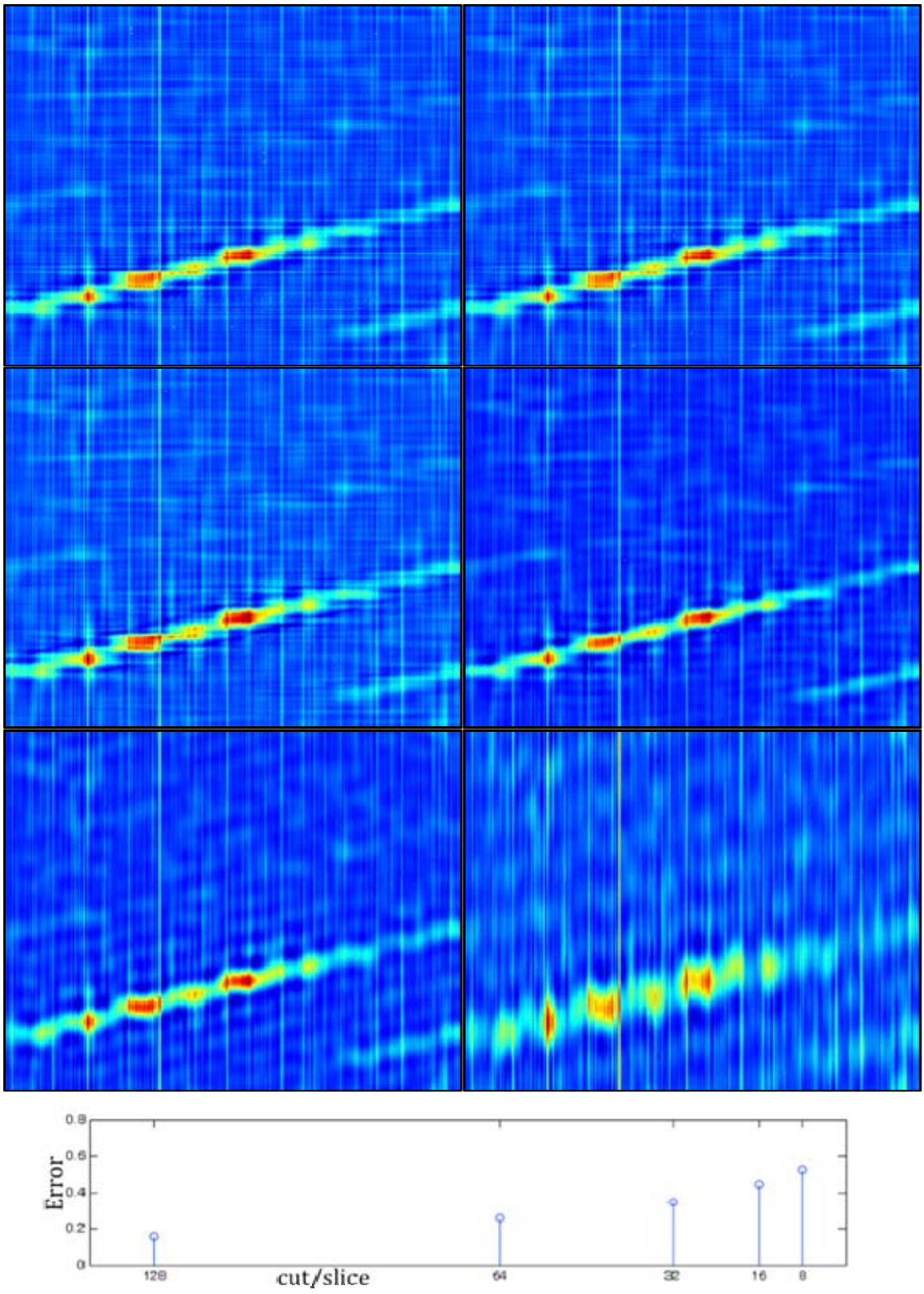


Figure 38. Cut and Slice results for P1, 0 dB, including a stem plot of the error when compared to the original (top left).

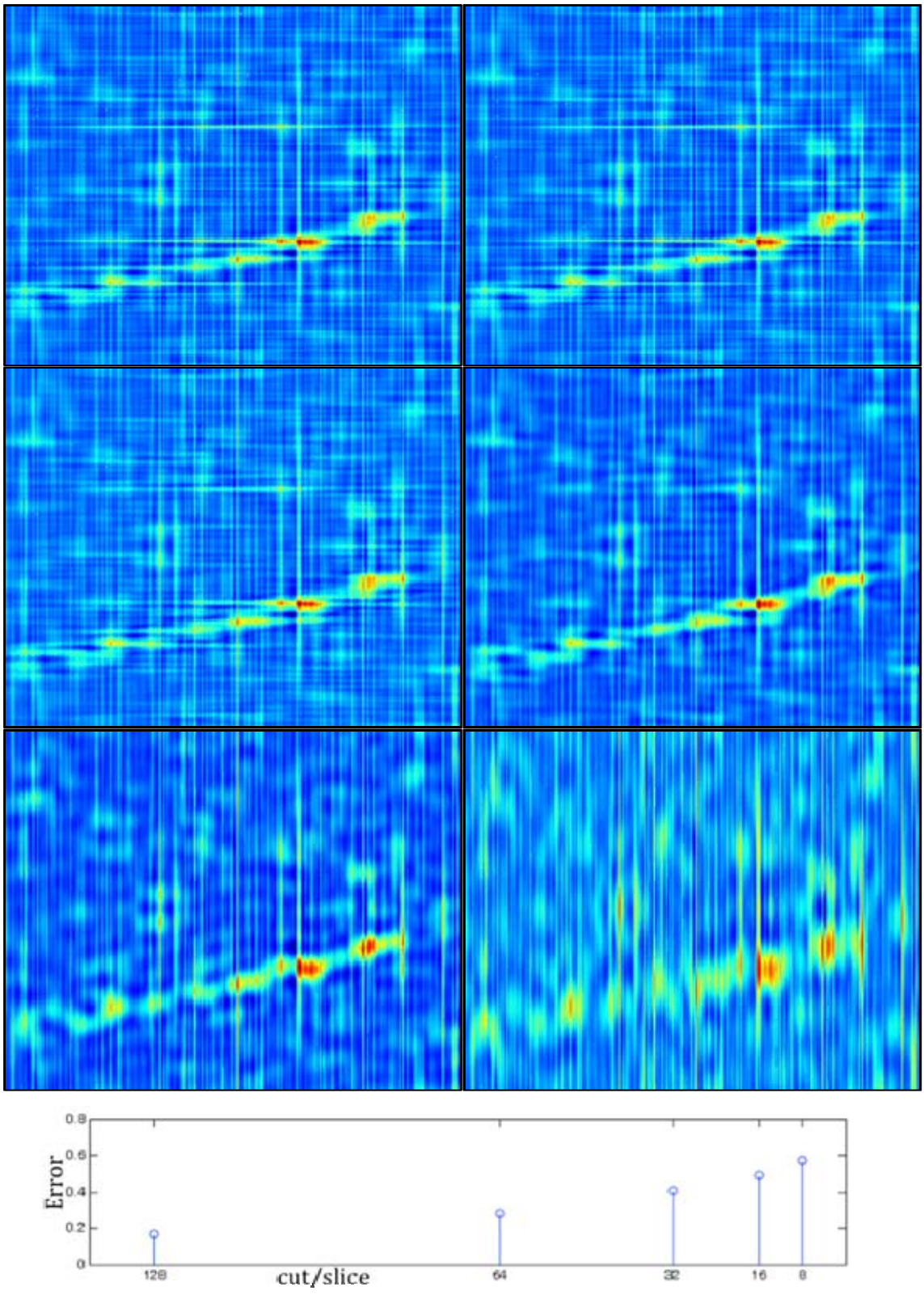


Figure 39. Cut and Slice results for P1, -6 dB, including a stem plot of the error when compared to the original (top left).

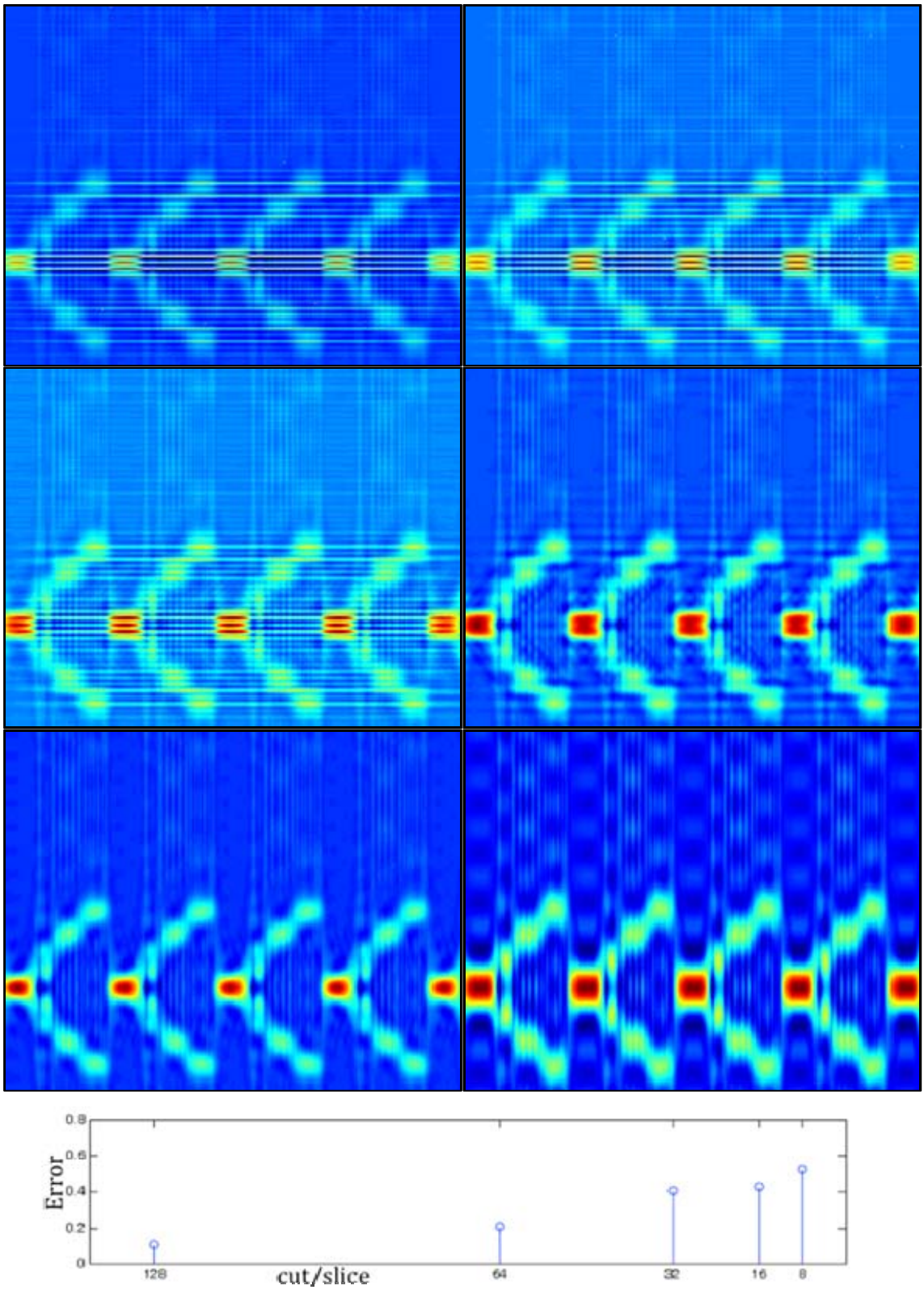


Figure 40. Cut and Slice results for PT1, signal only, including a stem plot of the error when compared to the original (top left).

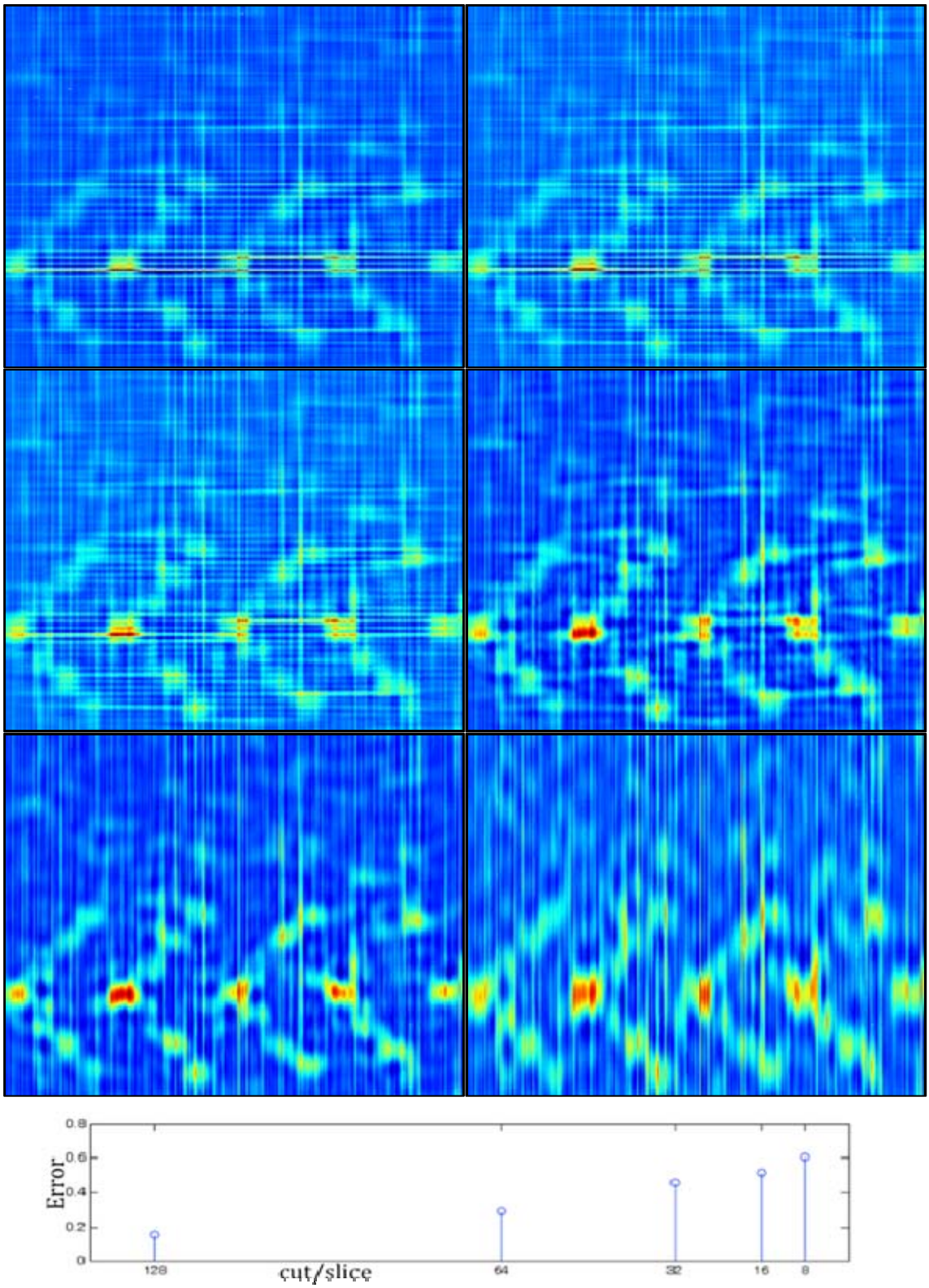


Figure 41. Cut and Slice results for PT1, 0 dB, including a stem plot of the error when compared to the original (top left).

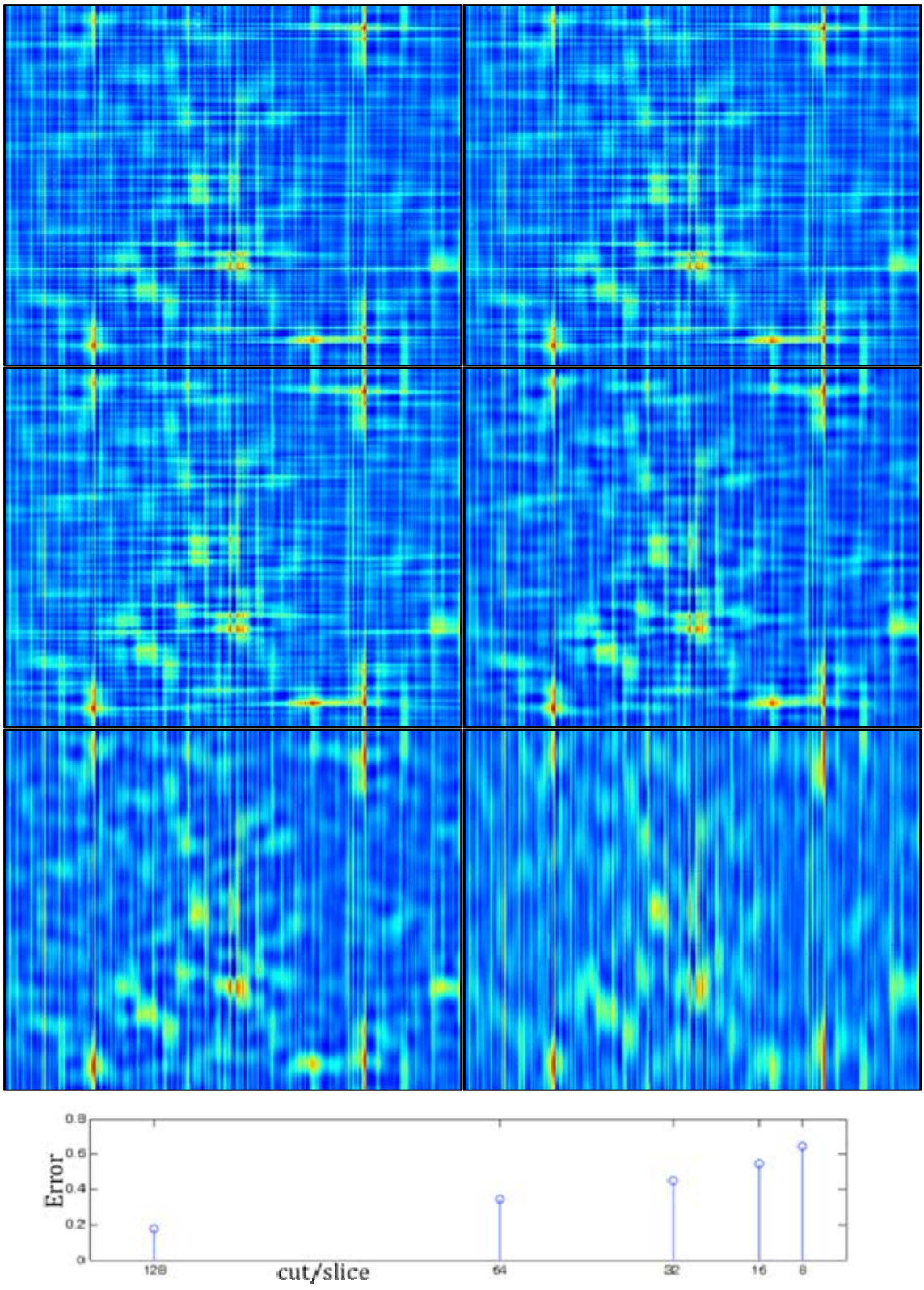


Figure 42. Cut and Slice results for PT1, -6 dB, including a stem plot of the error when compared to the original (top left).

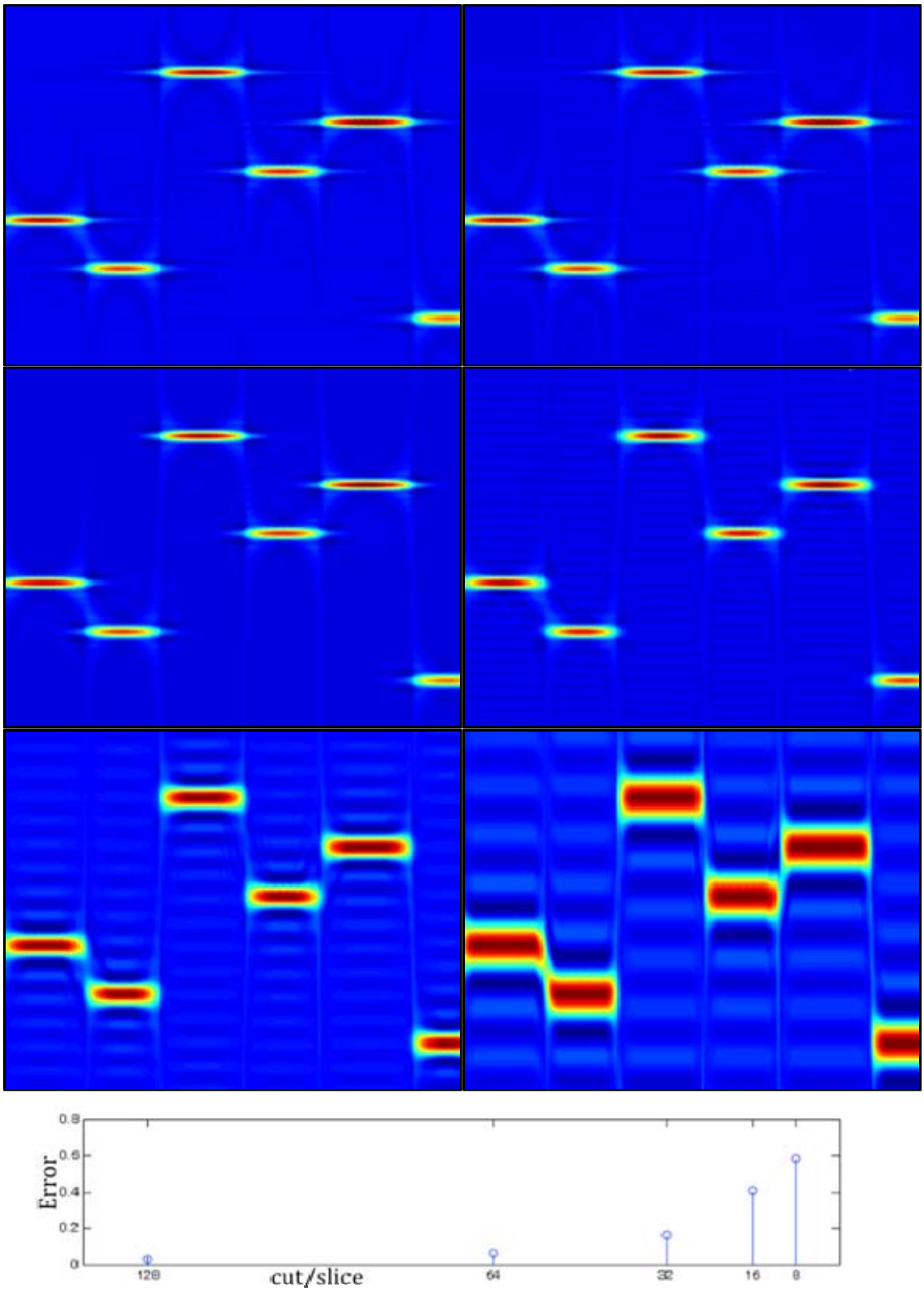


Figure 43. Cut and Slice results for Costas, signal only, including a stem plot of the error when compared to the original (top left).

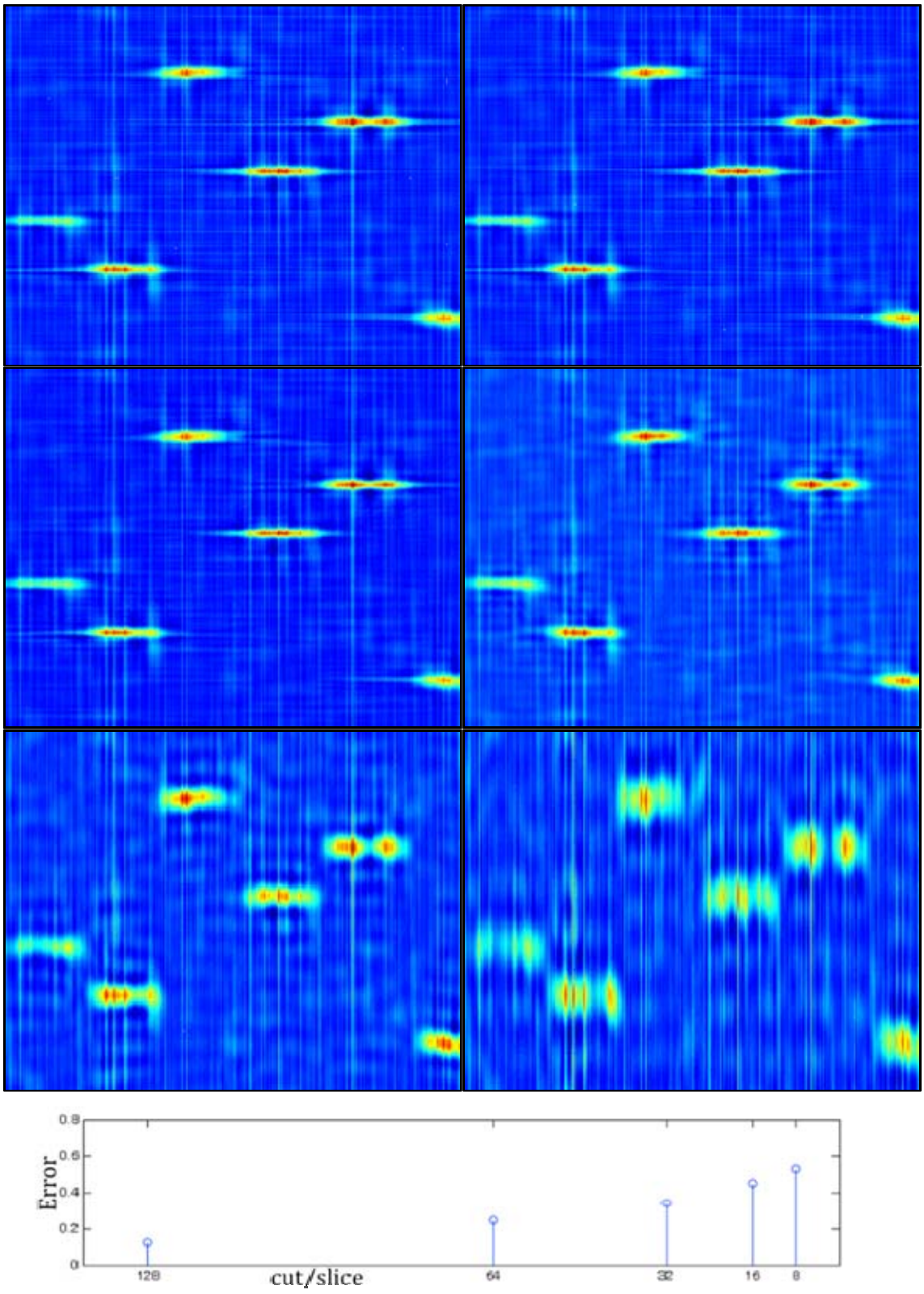


Figure 44. Cut and Slice results for Costas, 0 dB, including a stem plot of the error when compared to the original (top left).

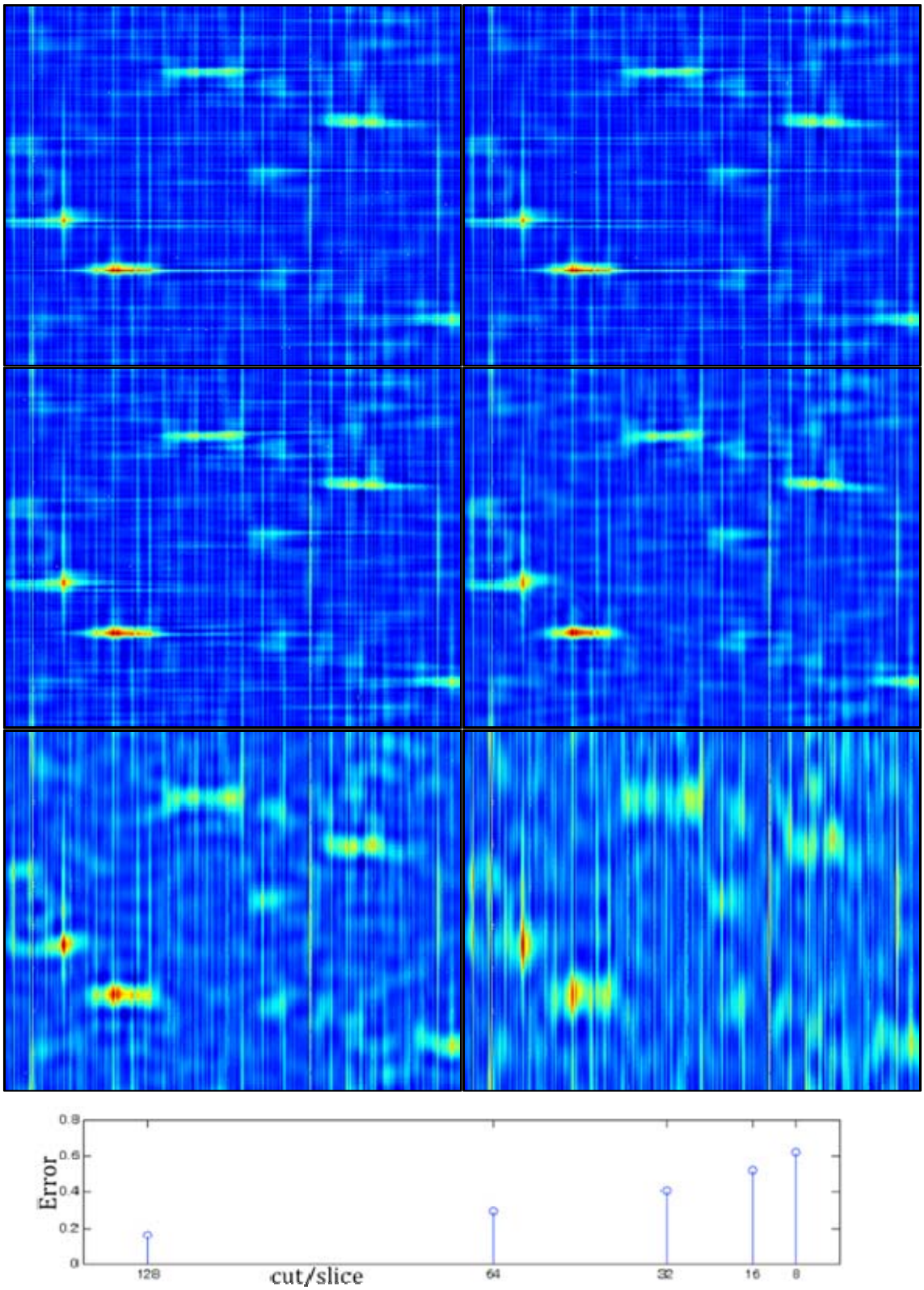


Figure 45. Cut and Slice results for Costas, -6 dB, including a stem plot of the error when compared to the original (top left).

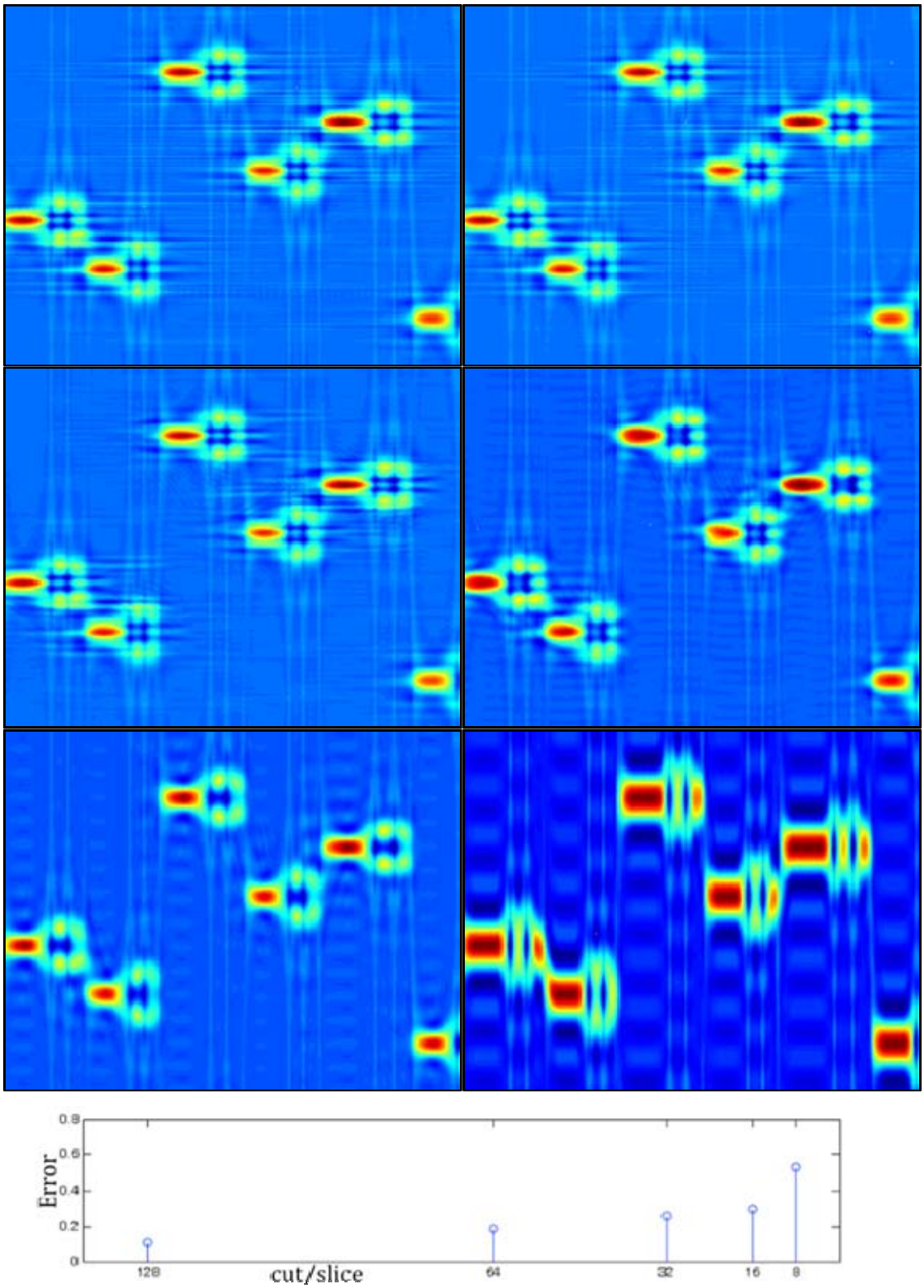


Figure 46. Cut and Slice results for FSK/PSK, signal only, including a stem plot of the error when compared to the original (top left).

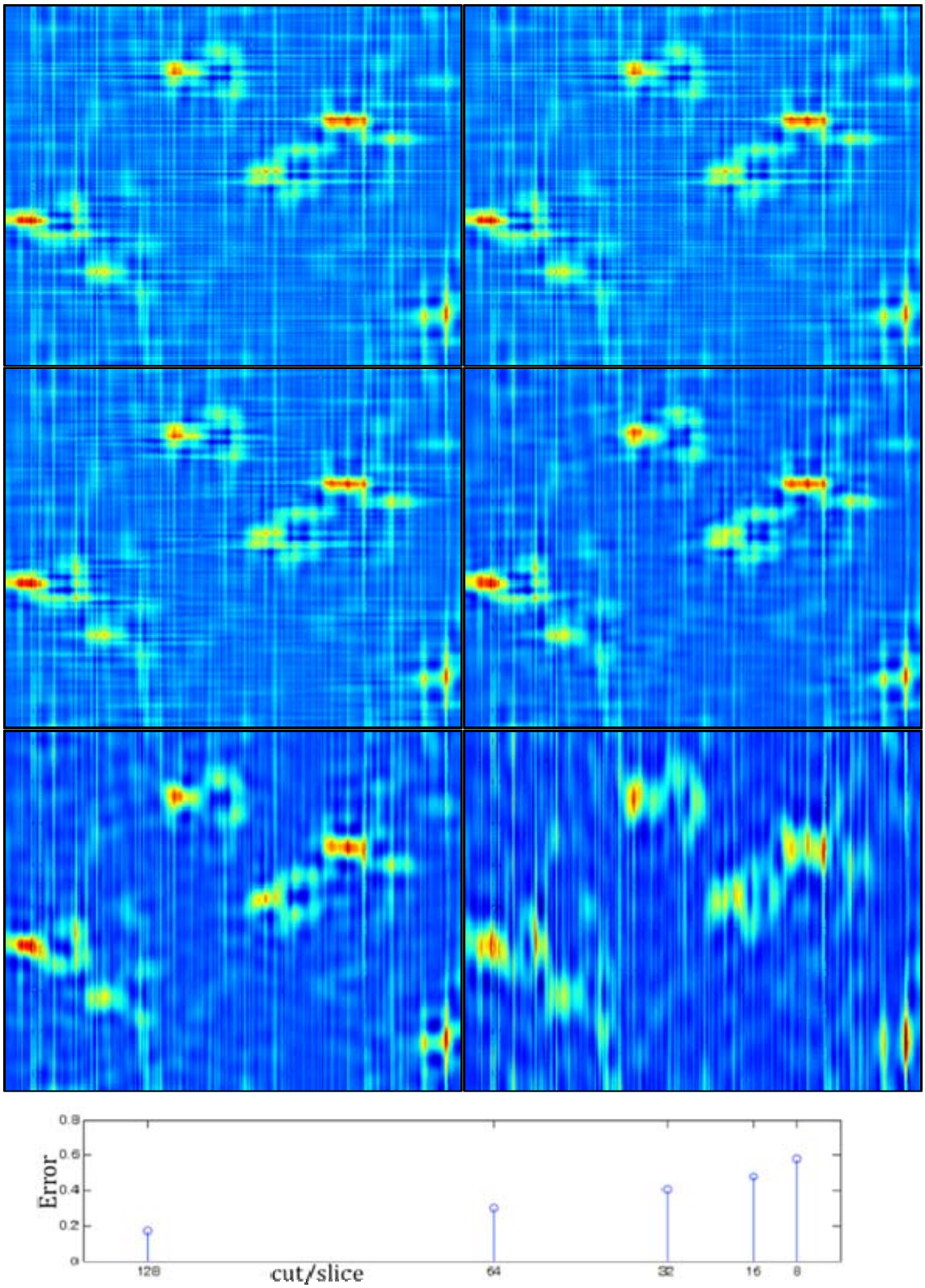


Figure 47. Cut and Slice results for FSK/PSK, 0 dB, including a stem plot of the error when compared to the original (top left).

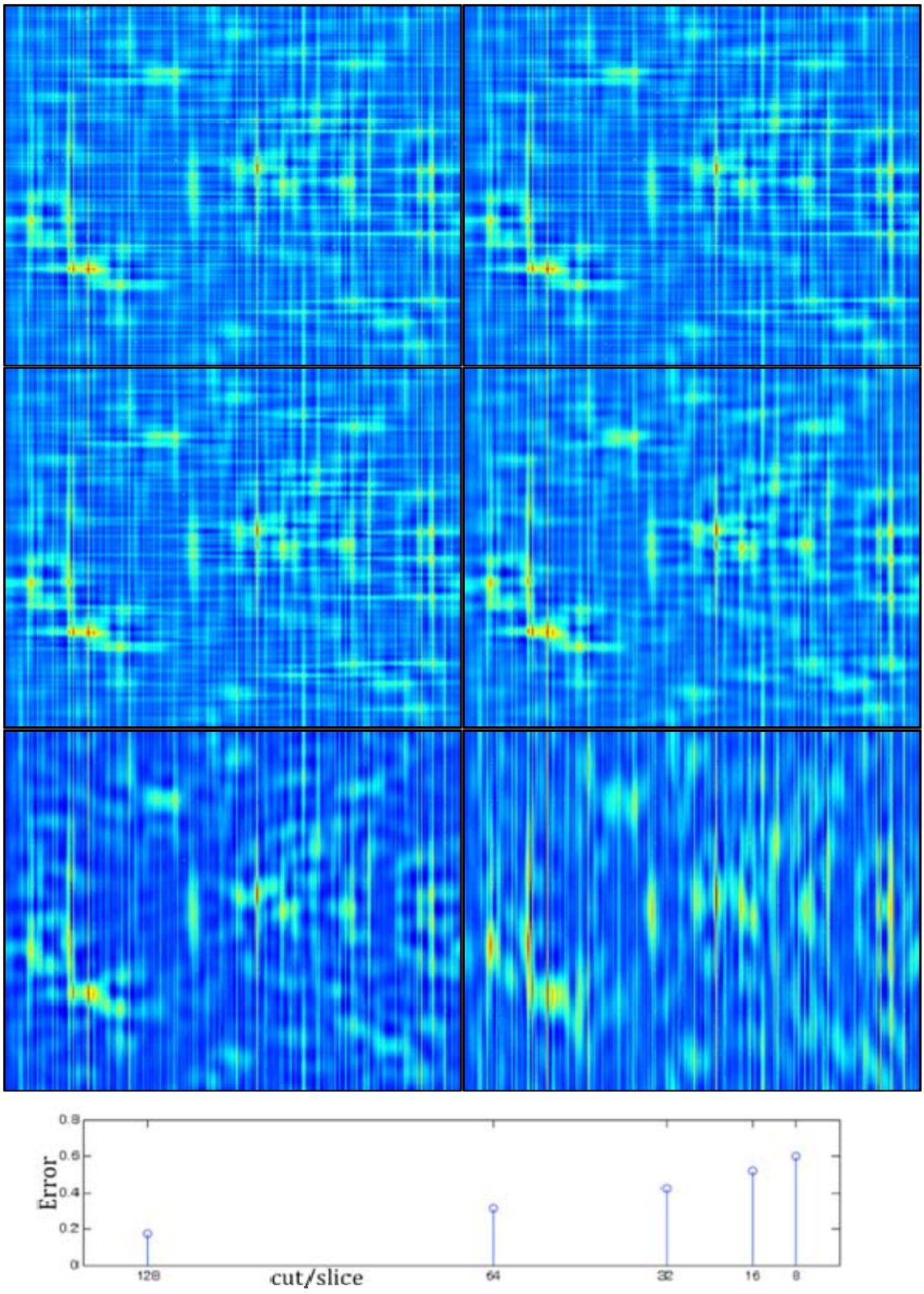


Figure 48. Cut and Slice results for FSK/PSK, -6 dB, including a stem plot of the error when compared to the original (top left).

As would be expected, in all cases in the above error plots, the more the kernel function was cut and sliced down, the more error was introduced (using the original distribution as the benchmark). Also, as would be hoped, the distribution for cut = slice = 32 picture of the signal was not degraded significantly. Interestingly, in many cases, the picture actually looks clearer. For example, note in Figure 49 that for the FMCW, signal only, the optimized version (for the rest of this thesis, optimized refers to the cut = slice = 32 version, and original refers to the un-optimized version of the distribution) looks much clearer than the original. If this is a signal only representation, why are there horizontal lines across the picture?

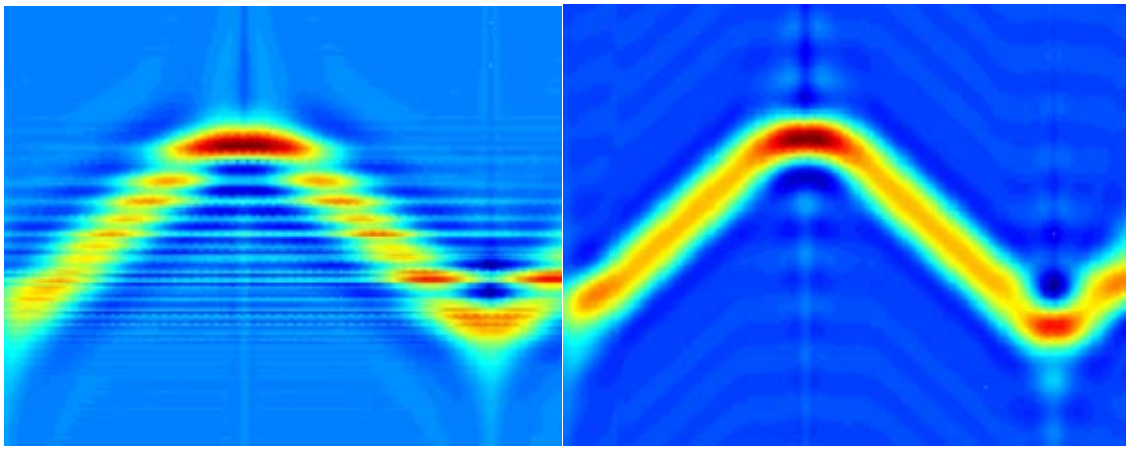


Figure 49. Original vs. Optimized FMCW, signal only.

A horizontal line across the plot indicates that that frequency is present throughout the entire sample. Figure 50 shows the kernel function for times $\ell = 0$ and $\ell = 200$. Note, that for low values of n (the columns on the left side), the Gaussian distribution has a low variance, but that at high values of n, the Gaussian distribution quickly approaches being completely flat, giving near equal representation to all values in the column.

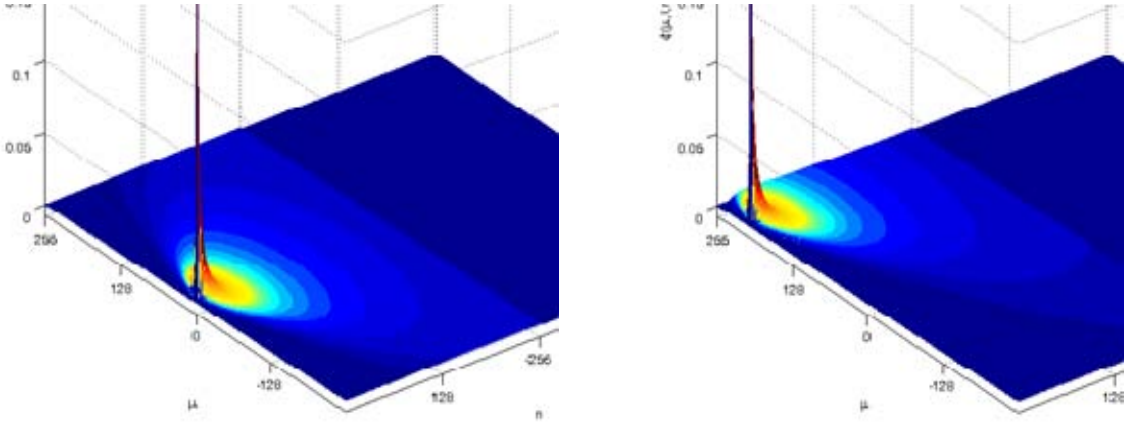


Figure 50. $\phi(\mu,\ell,n)$ for $\ell=0$ and $\ell=200$.

Figure 51 shows that $A(\mu,n)$ (for $N=8$), it can also be seen that for higher values of n, the column represents data points multiplied together that are more distant from each other in time.

$\mu \setminus n$	0	1	2	3	-4	-3	-2	-1
3	x(3)·x(3)*	0	0	0	0	0	0	0
2	x(2)·x(2)*	x(3)·x(1)*	0	0	0	0	0	x(1)·x(3)*
1	x(1)·x(1)*	x(2)·x(0)*	x(3)·x(-1)*	0	0	0	X(-1)·x(3)*	x(0)·x(2)*
0	x(0)·x(0)*	x(1)·x(-1)*	x(2)·x(-2)*	x(3)·x(-3)*	0	x(-3)·x(3)*	x(-2)·x(2)*	x(-1)·x(1)*
-1	x(-1)·x(-1)*	x(0)·x(-2)*	x(1)·x(-3)*	x(2)·x(-4)*	0	x(-4)·x(2)*	x(-3)·x(1)*	x(-2)·x(0)*
-2	x(-2)·x(-2)*	x(-1)·x(-3)*	x(0)·x(-4)*	0	0	0	x(-4)·x(0)*	x(-3)·x(-1)*
-3	x(-3)·x(-3)*	x(-2)·x(-4)*	0	0	0	0	0	x(-4)·x(-2)*
-4	x(-4)·x(-4)*	0	0	0	0	0	0	0

Figure 51. $A(\mu,n)$ for $N=8$.

Therefore, for higher instances of n, the summation over μ will provide near equal representation for all samples multiplied by another sample that is $2n$ in distance away—for all cases of ℓ.

Figure 52 shows the original FMCW, signal only, plot with the distributions for $\ell = 0$ and $\ell = 200$ highlighted in yellow. Figure 53 shows $S^H(\ell, n)$ for $\ell = 0$ and $\ell = 200$. Figure 54 shows a close up of the same plot for $200 \le n \le 250$. Figure 55 shows CWD for $\ell = 0$ and $\ell = 200$.

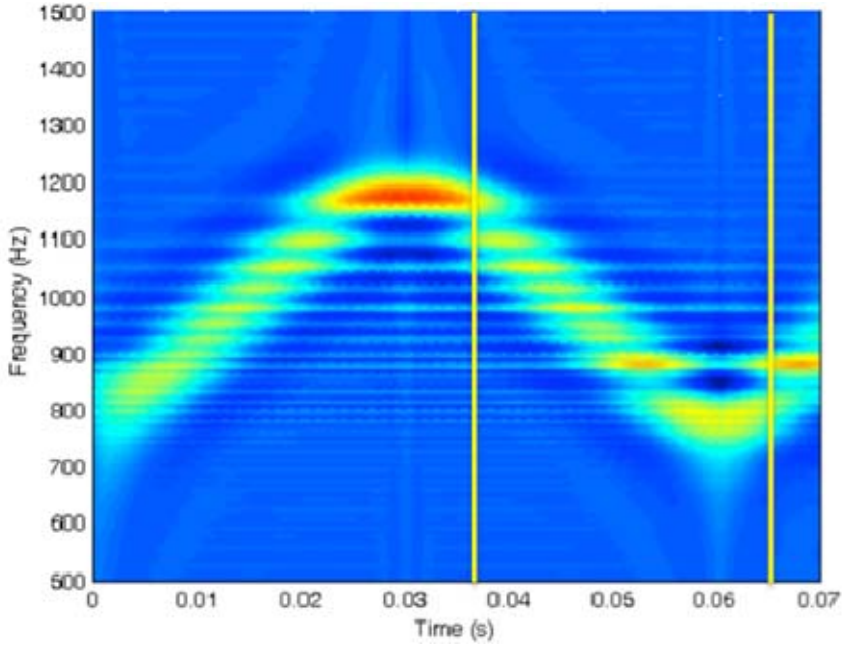


Figure 52. Highlighted original FMCW, signal only.

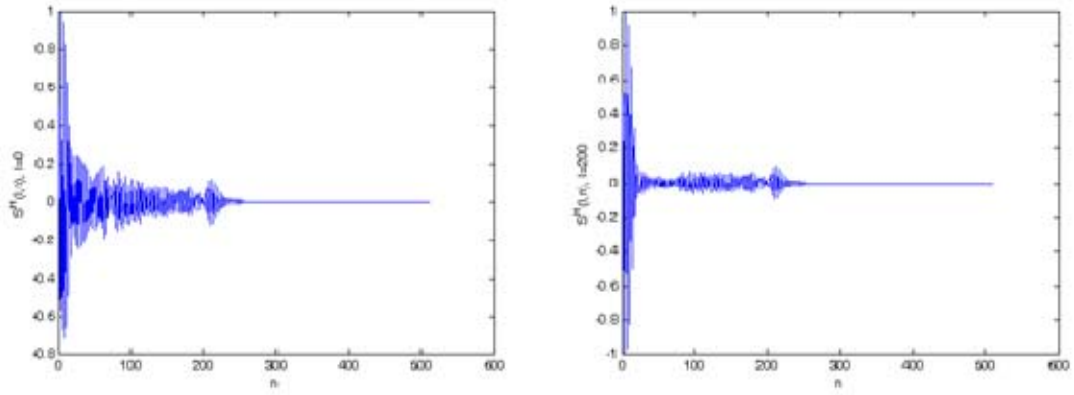


Figure 53. $S^H(\ell, n)$ for $\ell = 0$ and $\ell = 200$ for unaltered CWD.

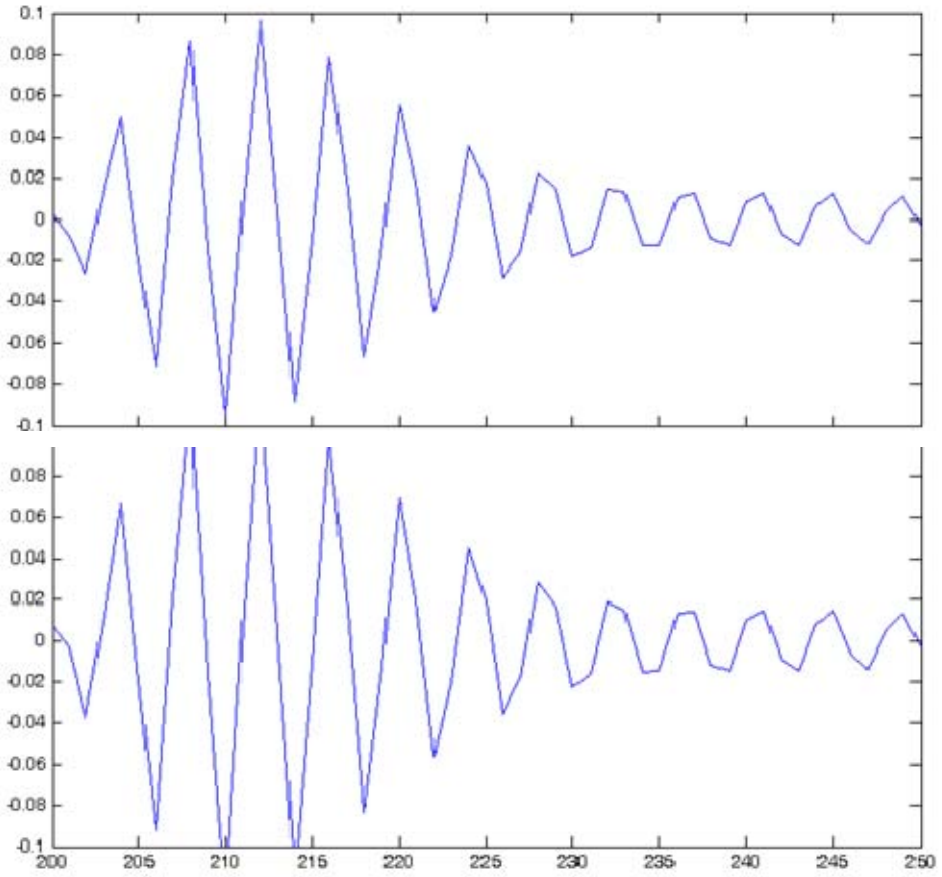


Figure 54. Close up of $S^H(\ell, n)$ for $\ell = 0$ and $\ell = 200$ for unaltered CWD.

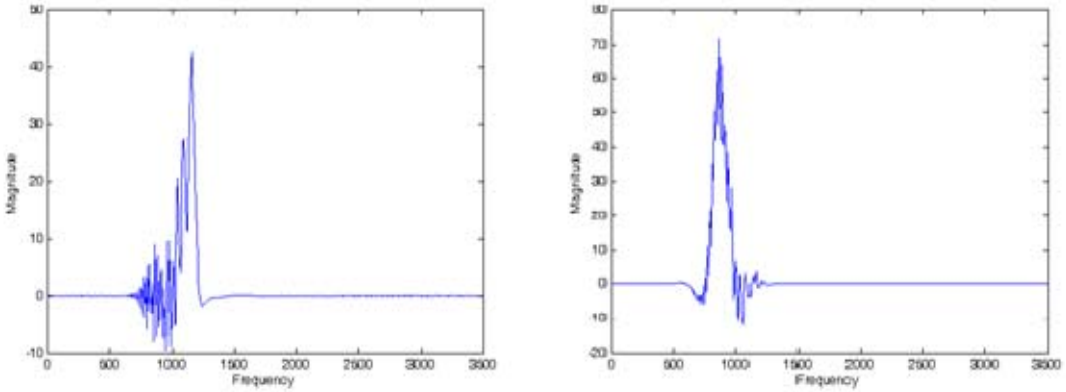


Figure 55. $CWD_x(\ell, \omega)$ for $\ell = 0$ and $\ell = 200$ for unaltered CWD.

Note, that as expected from the previous analysis, $S^H(\ell,n)$ produces nearly identical results for high values of n and highly disparate values of ℓ, and that these results are derived from samples multiplied by other samples that are $2n$ in distance away from each other. This is certainly not desired and is the cause of the horizontal lines across the original distribution. Figures 56 through 59 below conduct the same analysis for the optimized version of the distribution.

Figure 56 shows the optimized kernel function for times $\ell = 0$ and $\ell = 200$. Figure 57 shows the optimized CWD for the FMCW, signal only, plot with the distributions for $\ell = 0$ and $\ell = 200$ highlighted in yellow. Figure 58 shows $S^H(\ell,n)$ for $\ell = 0$ and $\ell = 200$. Finally, Figure 59 shows the optimized CWD for $\ell = 0$ and $\ell = 200$.

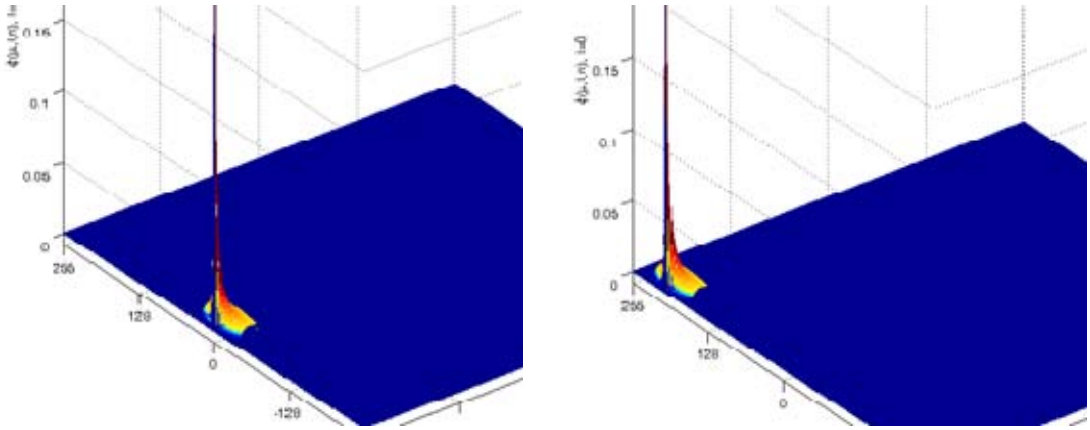


Figure 56. $\phi(\mu,\ell,n)$ for $\ell = 0$ and $\ell = 200$, cut = slice = 32.

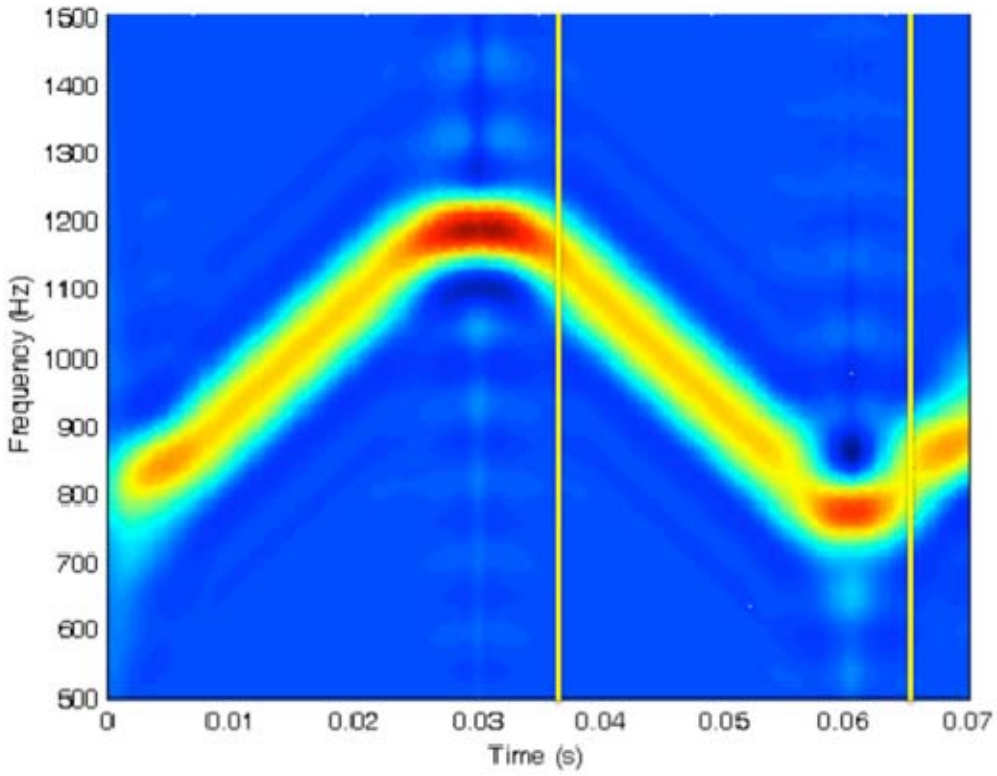


Figure 57. Highlighted optimized FMCW, signal only.

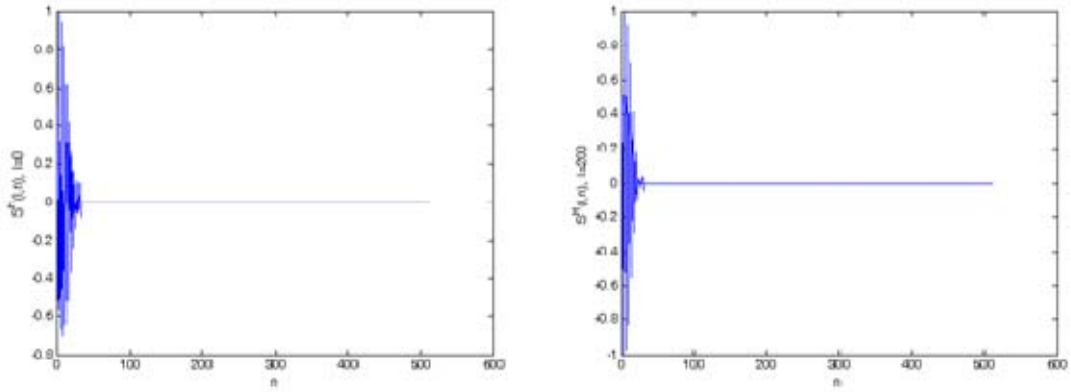


Figure 58. $S^H(\ell, n)$ for $\ell = 0$ and $\ell = 200$, cut = slice = 32.

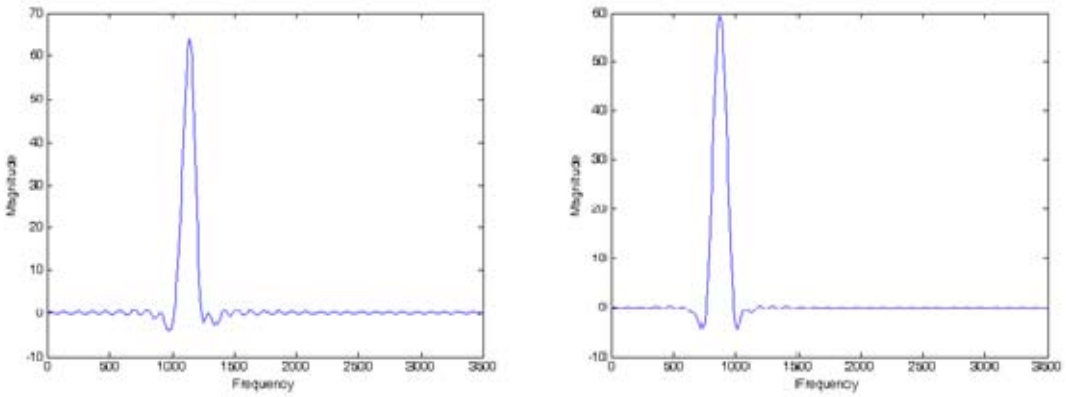


Figure 59. $CWD_x(\ell, \omega)$ for $\ell = 0$ and $\ell = 200$, cut = slice = 32.

Figure 60 shows three-dimensional representations of the CWD for both the original and optimized versions.

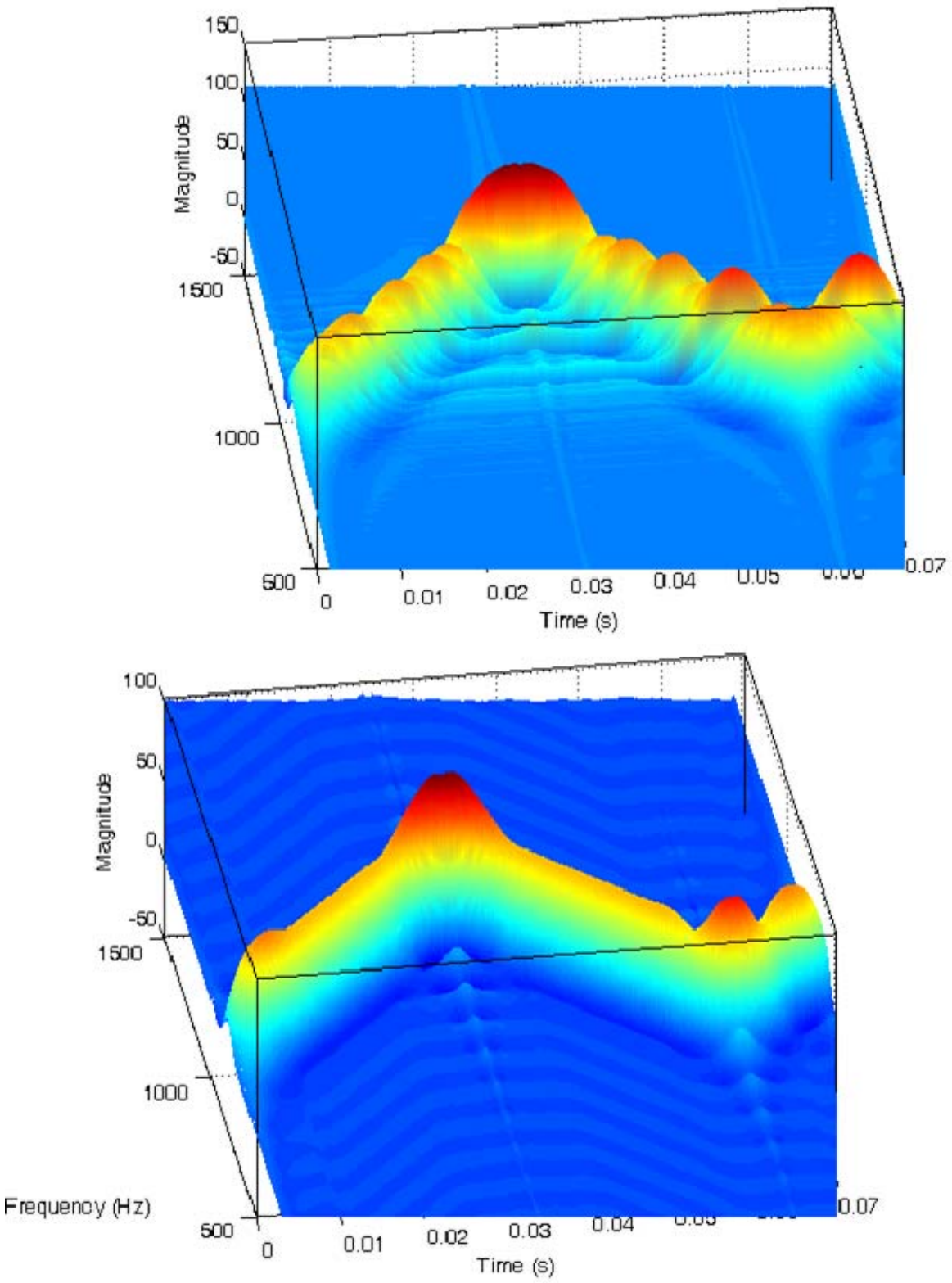


Figure 60. Original and Optimized FMCW, signal only.

It can be concluded from this analysis that not only is the optimized version faster and better looking, it portrays a more accurate depiction of the signal as well.

B. TIMING RESULTS

Coding of the optimized version of the algorithm was conducted incrementally. As each optimization was developed, it was verified using MATLAB to empirically determine that the math was correct, and then the optimizations were applied to the code in [5]. In comparison to the compiler un-optimized version of the original code, using the symmetry optimization yielded an approximately tenfold increase in speed. The cut and slice optimization yielded another approximately tenfold increase in speed. With the permission of Professor Breitenbach, the recursive FFT function authored by Professor Breitenbach used in the original code was unrolled, placed within the code itself and optimized in the same manner as described in the FFT optimization section (Chapter II), increasing the speed of computation by approximately another twofold. Other speed increases were realized by reducing redundant computations. For instance, in the original code, the kernel function is essentially recalculated N times. In the new code, the windowed (cut and sliced) kernel function is pre-calculated before the timing starts. The original code was also de-parameterized (the original was built to perform the distribution for any N where N is a power of two) in order to wring out every possible speed increase.

Finally, it made sense to pipeline the code. In the original algorithm, it was necessary to use all data samples for each and every iteration of ℓ, whereas for the optimized version, each iteration depends only on samples near in time to ℓ. In the final pipelined version, the arrays are preloaded with the samples needed to compute the first computation ($\ell = -256$), a new sample is written into the array over the oldest sample in the array, and the computation is redone. The priming of the pipeline before timing starts is cheating a little bit. However, it should only account for $1/512$th of the speed increase over the original code.

The pipelined version was developed with real, pipelined hardware in mind. The final code should be easy to port to the SRC-6 supercomputer or other high performance computing hardware.

To compare the final code (`pipe.c`) to the original code (`choi.c` from [5]) five test runs were conducted for both the compiler un-optimized and compiler-optimized versions of both programs. The compiler used was icc, and the optimizations used were:

-O3 –tpp7 –xW –align –Zp16 –ipo –static

The trial runs were conducted on an Intel chip, Linux based PC. Table 3 shows the results.

	From [5]		**From this work**	
	choi.c	**choi.c (compiler optimized)**	**pipe.c**	**pipe.c (compiler optimized)**
Trail 1	46.81	6.860	0.05442	0.04699
Trial 2	46.51	6.887	0.05445	0.04655
Trial 3	46.50	6.852	0.05457	0.04640
Trial 4	46.23	6.872	0.05470	0.04631
Trial 5	46.49	6.824	0.05426	0.04661
Average	**46.51**	**6.859**	**.05448**	**.046572**

Table 3. Time in seconds of trial runs.

The optimized version of the code produced an 854X increase in speed over the original code. The optimized version of the code compiled with an optimizing compiler produced a 147X increase in speed over the compiler original version of the code compiled with an optimizing compiler.

THIS PAGE INTENTIONALLY LEFT BANK

V. CONCLUSION

A. BACKGROUND

The objective of this thesis was to improve the speed at which the CWD could be computed on the SRC-6 reconfigurable supercomputer. Optimizing the algorithm, prior to porting the code to the SRC-6, yielded enough work in both quantity and level of difficulty to stand alone as the subject of this thesis. Also, the optimizations developed in this thesis are applicable to any implementation of the CWD.

B. RESULTS

By exploiting the symmetry of the CWD and eliminating the computation of near zero terms, dramatic gains in computation speed were achieved. Further gains were achieved by modifying the FFT to take advantage of zero terms. The optimizations altered the results of the time-frequency distribution; however, the altered results yield a more accurate time-frequency representation of the signal. The optimized algorithm developed in this thesis is a significant step towards developing a system that can identify and classify LPI signals in real time. This algorithm is not platform specific, since developed using pure Mathematics. It can be used to realize faster Choi-Williams calculations, with a better result, on any platform.

C. RECOMMENDATIONS FOR FUTURE WORK

It is recommended that the effort to port this algorithm to the SRC-6 or some other FPGA realization be continued. Particularly, the “Optimization Not Realized” from Chapter III should yield a significant increase in speed if implemented using an FPGA.

THIS PAGE INTENTIONALLY LEFT BANK

APPENDIX. LPI SIGNAL GENERATION

The example signals used for this thesis were taken from reference [14]. There were several signals generated by the LPI Toolbox used in this thesis. Each signal is described in detail below. Each signal was generated with a signal to noise ratio of 0 dB. The addition of noise generates more realistic results without overwhelming the graph.

1) F_1_7_500_30_s.mat: FMCW signal with a ∞ dB SNR
2) F_1_7_500_30_0.mat: FMCW signal with a 0 dB SNR
3) F_1_7_500_30_-6.mat: FMCW signal with a -6 dB SNR
4) P1_1_7_8_1_s.mat: P1 signal with a ∞ dB SNR
5) P1_1_7_8_1_0.mat: P1 signal with a 0 dB SNR
6) P1_1_7_8_1_-6.mat: P1 signal with a -6 dB SNR
7) PT1_1_7_2_4_s.mat: PT1 signal with a ∞ dB SNR
8) PT1_1_7_2_4_0.mat: PT1 signal with a 0 dB SNR
9) PT1_1_7_2_4_-6.mat: PT1 signal with a -6 dB SNR
10) C_1_15_5000_s.mat: Costas signal with a ∞ dB SNR
11) C_1_15_5000_0.mat: Costas signal with a 0 dB SNR
12) C_1_15_5000_-6.mat: Costas signal with a -6 dB SNR
13) FSK_PSK_Costas_5_s.mat: FSK/PSK signal with a ∞ dB SNR
14) FSK_PSK_Costas_5_0.mat: FSK/PSK signal with a 0 dB SNR
15) FSK_PSK_Costas_5_-6.mat: FSK/PSK signal with a -6 dB SNR

More information on the use of the LPI Toolbox and the different LPI signals is given in [4].

To use these signals in the C programming environment, they were converted to text files using the following code [5], [14].

```
%% %%%%%%%%%%%%%%%%%%%% FMCW Code %%%%%%%%%%%%%%%%%%%% %%
load H:\Thesis\Choi\Test_signals\F_1_7_250_20_0.mat
sig1 = [I Q];
save S:\thesis\Test_signals_txt\F_1_7_250_20_0.txt sig1 -ascii -double

%% %%%%%%%%%%%%%%%%%%%% Frank Code %%%%%%%%%%%%%%%%%%%% %%
load H:\Thesis\Choi\Test_signals\FR_1_7_4_1_0.mat
sig2 = [I Q];
save S:\thesis\Test_signals_txt\FR_1_7_4_1_0.txt sig2 -ascii -double

%% %%%%%%%%%%%%%%%%%%%% Costas Code %%%%%%%%%%%%%%%%%%%% %%
load H:\Thesis\Choi\Test_signals\C_1_15_5000_0.mat
sig3 = [I Q];
save S:\thesis\Test_signals_txt\C_1_15_5000_0.txt sig3 -ascii -double

%% %%%%%%%%%%%%%%%% FSK/PSK Costas Code %%%%%%%%%%%%%%%% %%
load H:\Thesis\Choi\Test_signals\FSK_PSK_Costas_15_5_0.mat
sig4 = [I Q];
save S:\thesis\Test_signals_txt\FSK_PSK_Costas_15_5_0.txt sig4 -ascii
        -double
```

LIST OF REFERENCES

[1] H. I. Choi and W. J. Williams, “Improved time-frequency representation of multicomponent signals using exponential kernels,” *IEEE trans. Acoust., Speech, Signal Processing,* vol. 37, no. 6, pp. 862–871, 1989.

[2] L. Cohen, “Time-frequency distributions – A review,” *Proc. IEEE,* vol. 77, no 7, pp 941–981, 1989.

[3] B. Boasash and P. J. Black, “An efficient real-time implementation of the Wigner-Ville distribution,” *IEEE trans. Acoust., Speech, Signal Processing,* vol. 35, pp. 1611–1618, 1987.

[4] G. J. Upperman, T.L.O.Upperman, D. Fouts, and P. Pace, “Efficient time-frequency and bi-frequency signal processing on a reconfigurable computer,” *IEEE Asilomar Conf.,* pp. 176–180, 2008.

[5] T.L. Upperman, “ELINT Signal Processing Using Choi-Wiliiams Distribution on Reconfigurable Computers for Detection and Classification of LPI Emitters,” M.S. thesis, Dept. Electrical and Computer Eng., Naval Postgraduate School, Monterey, CA, 2008.

[6] J. Cardoso, P. Fish, and M. Ruano, “Parallel implementation of a Choi-Williams TFD for Doppler signal analysis”, *IEEE proc. 20th conf. Eng. In Med. and Bio. Soc.,* vol. 20, no 3, 1998.

[7] D. T. Barry, “Fast Calculation of the Choi-Williams Time-Frequency Distribution,” in *IEEE trans. Signal Processing*, vol. 40, no. 2, Feb 1992, pp. 450–455

[8] G. Jones, and B. Boasash, “Instantaneous frequency, instantaneous bandwidth and the analysis of multicomponent signals,” *IEEE trans. Acoust., Speech, Signal Processing,* vol. 5, pp. 2467–2470, 1990.

[9] S. Narayanan, and K. Prabhu, “New method of computing Wigner-Ville distribution”, *Electronics Letters,* vol. 25, no 5, pp. 336–338, 2 March 1989.

[10] D. Jones and T. Parks, “A resolution comparison of several time-frequency representations,” *IEEE trans. Signal Processing,* vol. 40, no. 2, pp. 413–420, Feb 1992.

[11] G. Cunningham, and W. Williams, “Kernel decomposition of time-frequency distributions,” *IEEE trans. Signal Processing,* vol. 42, no. 6, pp. 1425–1442, June 1994.

[12] U.S. Department of Commerce, *Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables*, National Bureau of Standards, Applied Mathematics Series-55, Issued June 1964, Seventh Printing, May 1968, with corrections.

[13] Oppenheim & Schafer, *Digital Signal Processing,* Prentice-Hall, 1975.

[14] P. E. Pace, *Detecting and Classifying Low Probability of Intercept Radar*, 2ns Edition, Artech House, Inc., Norwood, MA, 2009.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

4. Director, Training and Education, MCCDC, Code C46
   Quantico, Virginia

5. Director, Marine Corps Research Center, MCCDC, Code C40RC
   Quantico, Virginia

6. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
   Camp Pendleton, California

7. Chairman, Code EC
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

8. Douglas J. Fouts
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

9. Phillip E. Pace
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

10. Jon Butler
    Department of Electrical and Computer Engineering
    Naval Postgraduate School
    Monterey, California

11. Jerome Breitenbach
Electrical Engineering Department
Cal Poly State University
San Luis Obispo, California

12. Alfred Di Mattesa
Naval Research Laboratory
Code 5701
Washington, D.C.

13. Peter Craig
Office of Naval Research
Code 312
Washington, D.C.

14. Jerry Fudge
Integrated Systems
L-3 Communication Systems
Greenville, Texas

15. Frank Boyle
Integrated Systems
L-3 Communication Systems
Greenville, Texas